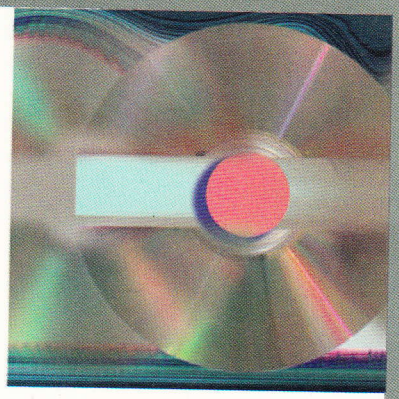
MINISTERUL EDUCAȚIEI ŞI CERCETĂRII



Informatică

Manual pentru clasa a XI-a

Mioara Gheorghe
(coordonator)
Monica Tătărâm
Corina Achinca
Constanța Năstase

CORINT

Capitolul 1

STRUCTURI DE DATE

1. ORGANIZAREA DATELOR

STUDIU DE CAZ ***Compania Eficient***

Pentru reducerea cheltuielilor cu amenajarea spațiilor comerciale, reclamă și personal, compania Eficient selectează tineri distribuitori, absolvenți de liceu. Oferta companiei este foarte atrăgătoare; de aceea, numărul solicitanților depășește numărul de posturi oferite (n). Selecția candidaților se face în ordinea înregistrării scrisorii de intenție și a CV-ului. Dosarele candidaților sunt păstrate într-un fișet, unul peste altul, în ordinea angajării. Periodic, compania trimite un angajat la cursuri de formare; este trimis întotdeauna, ultimul angajat (fig. 1). Angajații sunt plătiți în funcție de numărul de produse distribuite (vândute) zilnic. Săptămânal, managerul companiei ține evidența pe zile a produselor distribuite de fiecare angajat. În orice moment, managerul poate determina angajatul cu cea mai bună activitate într-o zi sau ziua în care a fost distribuit cel mai mic număr de produse. La sfârșitul săptămânii, după ce aplică bonusuri sau penalizări, managerul centralizează aceste date într-un registru de evidență a vânzărilor realizate de către fiecare angajat.

Managerul companiei dorește să prelucreze cu calculatorul datele pentru selecția candidaților, evidența angajaților și evidența vânzărilor.

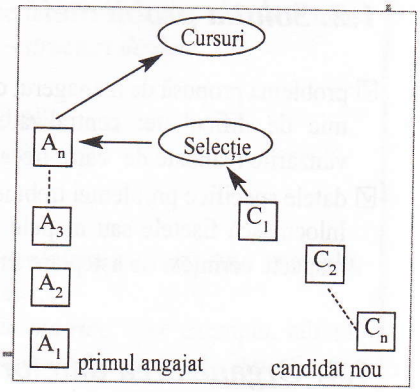


Figura 1

1.1. Analiza problemei

☑ datele despre candidați; din CV-ul fiecărui candidat vor fi reținute următoarele informații:
- numele,
- anul nașterii,
- media la examenul de bacalaureat;

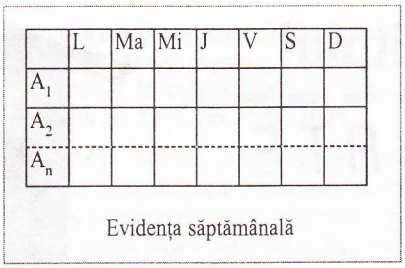


Figura 2

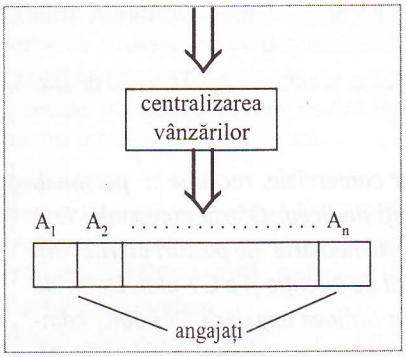


Figura 3

☑ selecția candidaților se face în ordinea înregistrării datelor personale;

☑ după examinarea unui candidat (selecție), datele acestuia nu mai sunt necesare; va intra în selecție candidatul următor;

☑ întotdeauna, datele unui nou candidat sunt așezate după datele ultimului candidat care așteaptă pentru selecție;

☑ datele angajaților sunt păstrate într-o ordine care să permită numirea rapidă a ultimului angajat în vederea trimiterii sale la cursuri;

☑ pentru evidența săptămânală a vânzărilor, se reține numărul de produse distribuite (vândute) zilnic de către fiecare angajat (fig. 2);

☑ pentru evidența centralizată a vânzărilor, totalul vânzărilor realizate într-o săptămână de fiecare angajat se adaugă la vânzările realizate de acel agajat până la momentul respectiv (fig. 3).

1.2. Soluția problemei

☑ problema propusă de managerul companiei Eficient necesită prelucrări matematice cu un grad mic de dificultate: centralizarea săptămânală, prin însumare, a valorilor reprezentând vânzările realizate de către fiecare angajat;

☑ datele specifice problemei trebuie organizate avantajos, astfel încât folosirea calculatorului să înlocuiască fișetele sau mapele în care sunt păstrate dosarele candidaților/angajaților și să respecte cerințele de așteptare sau prioritate specifice problemei.

1.3. Organizarea datelor

• Din analiza problemei urmărind *semnificația și complexitatea datelor*, rezultă două categorii de date:

☒ *date elementare*, spre exemplu: numărul de produse vândute de un angajat la un moment dat;

☒ *date grupate (structurate):*

– *date cu aceeași semnificație* (de același tip) – numite și *date omogene* –, spre exemplu: evidența săptămânală a vânzărilor pe zile și angajați, evidența vânzărilor pe angajați, datele despre toți angajații/candidații;

– *date cu semnificații diferite* – numite și *date neomegene* – spre exemplu, datele prin care este descrisă o persoană (candidat sau angajat): nume, an, medie.

De reținut!

O persoană este descrisă prin mai multe tipuri de date, ceea ce determină aspectul neomogen al grupului de date.

Candidații sau angajații formează grupuri de același tip – *persoană* – de unde rezultă aspectul omogen al acestui grup de date.

- Din analiza problemei urmărind *restricțiile de intrare-ieșire* (așteptare sau prioritate), după care sunt organizate datele dintr-un grup (structură), rezultă două categorii de date:

☒ date organizate după disciplina specifică unei *cozi* sau *fir de așteptare*; spre exemplu, candidații care așteaptă pentru selecție. Într-o astfel de structură, întotdeauna, un element nou este așezat (intră) după ultimul element. Dintr-o astfel de structură, iese, întotdeauna, primul element – în exemplul nostru, candidatul aflat „la rând".

☒ date organizate după disciplina specifică așezării obiectelor unul peste altul, în *stivă*; spre exemplu, dosarele angajaților. Într-o astfel de structură, întotdeauna, un element nou este așezat deasupra celorlalte, în *vârful* stivei. Dintr-o astfel de structură, iese, întotdeauna, elementul din *vârful* stivei – în exemplul nostru, va fi prelucrat dosarul ultimului angajat.

Concluzie

- organizarea datelor specifice unei probleme se poate face în locații de memorie independente – *date elementare* – sau în locații grupate – *structuri de date;*
- structurile de date pot fi:
  – *structuri omogene* (date cu aceeași semnificație/tip),
  – *structuri neomegene* (date cu semnificații/tipuri diferite);
- structurile omogene se numesc *tablouri;*
- într-un tablou, datele pot fi organizate astfel:
  – după un singur criteriu – *tablouri unidimensionale* sau *vectori*; spre exemplu, tabloul pentru evidența vânzărilor realizate de fiecare angajat,
  – după două criterii – *tablouri bidimensionale* sau *matrice*; spre exemplu, tabloul pentru evidența vânzărilor realizate zilnic (criteriul 1 – zilele săptămânii) de către fiecare angajat (criteriul 2 – angajații);
- într-un tablou unidimensional, datele pot fi organizate astfel încât să respecte o disciplină de intrare/ieșire de tip *coadă* sau *stivă*;
- structurile neomogene se numesc *articole* sau *înregistrări*; mai multe articole care descriu aceeași entitate (obiect, persoană etc.) reprezintă un grup de date cu aceeași semnificație și poate fi organizat într-o structură omogenă de tip tablou unidimensional (*vector de articole*).

TEME

1. Explicați deosebirea dintre analiza datelor din punct de vedere al complexității și analiza datelor din punct de vedere al regulilor de organizare (intrare/ieșire) într-o structură (grup de date).
2. Formulați un exemplu care să evidențieze diferența dintre datele omogene și datele neomegene.
3. Formulați un exemplu care să necesite organizarea după mai multe criterii a datelor cu aceeași semnificație.
4. Formulați un exemplu care să necesite organizarea datelor cu aceeași semnificație în structuri de tip *coadă*.
5. Formulați un exemplu care să necesite organizarea datelor cu aceeași semnificație în structuri de tip *stivă*.
6. Explicați disciplina structurii de date de tip coadă (FIFO – First Input First Output = *primul intrat, primul ieșit*).
7. Explicați disciplina structurii de date de tip stivă (LIFO – Last Input First Output = *ultimul intrat, primul ieșit).*
8. Formulați exemple de oganizare a datelor după alte reguli decât cele specifice structurilor de tip *coadă* sau *stivă*.

TEMĂ de GRUP

Densitatea straturilor geologice

Un grup de geologi studiază densitatea straturilor geologice. În acest scop, s-a extras câte o probă pentru fiecare dintre cele ***n*** *straturi geologice studiate. Fiecare probă este transmisă la un laborator specializat; pentru fiecare probă se fac mai multe teste, cel mult* ***m****. Întrucât nu există suficiente aparate, analiza de laborator durează. La sfârșitul activității, geologii păstrează probele în ordinea naturlă a straturilor geologice* (fig. 4)*. Rezultatele testelor se păstrează astfel încât să se poată determina, cu ușurință:*

– zăcământul la care s-au obținut aceleași valori la toate testele;

– zăcământul cu cea mai mare densitate medie la cele ***m*** *teste;*

– zăcământul la care s-a obținut cea mai mare diferență între densitatea maximă și densitatea minimă din cele ***m*** *teste.*

Cerințe:

1. Identificați formele de organizare a datelor specifice problemei propuse.
2. Fiecare membru al grupului descrie una dintre formele de organizare a datelor identificate (justificare, necesitate, proprietăți, prelucrări specifice și alte aspecte).

3. Grupul compune un scenariu pentru prelucrarea cu calculatorul a datelor specifice acestei probleme; scenariul poate fi prezentat narativ sau organizat pe secvențe (pași).

4. Grupul realizează o prezentare PowerPoint care să ajute la susținerea temei.

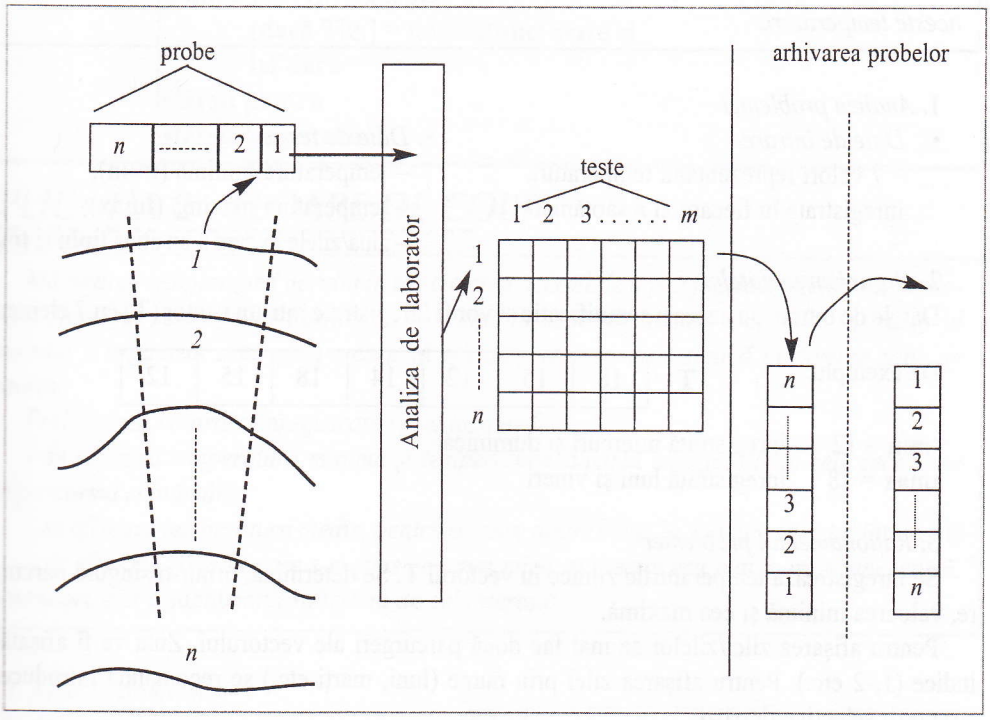


Figura 4

SUGESTIE DE PREZENTARE:

Prezentările pot fi expuse și analizate în laboratorul de Informatică; se va urmări corectitudinea rezolvării, creativitatea prezentării și eficiența lucrului în grup.

2. ORGANIZAREA DATELOR CU ACEEAȘI SEMNIFICAȚIE ÎN TABLOURI BIDIMENSIONALE

APLICAȚIA 1 FLOARE DE COLȚ

Membrii clubului „Floare de colț" participă la o tabără de vară, timp de o săptămână, într-o zonă montană greu accesibilă. Scopul lor este să înregistreze în fiecare zi temperatura aerului și să determine temperatura minimă, temperatura maximă și zilele în care s-au atins aceste temperaturi.

1. *Analiza problemei*

- *Date de intrare*
  – 7 valori reprezentând temperatura înregistrată în fiecare zi a săptămânii
- *Date de ieșire*
  – temperatura minimă (**tmin**);
  – temperatura maximă (**tmax**);
  – ziua/zilele în care s-au atins **tmin** și **tmax.**

2. *Organizarea datelor*

Datele de intrare au aceeași semnificație și vor fi înregistrate într-un vector (**T**) cu 7 elemente.

De exemplu:

T	18	15	12	14	18	15	12

tmin = 12 înregistrată miercuri și duminică
tmax = 18 înregistrată luni și vineri

3. *Raționamentul problemei*

Se înregistrează temperaturile zilnice în vectorul **T**. Se determină, printr-o singură parcurgere, valoarea minimă și cea maximă.

Pentru afișarea zilei/zilelor se mai fac două parcurgeri ale vectorului. Ziua va fi afișată ca indice (1, 2 etc.). Pentru afișarea zilei prin nume (luni, marți etc.) se recomandă introducerea structurii selective după **zi**.

4. *Reprezentarea algoritmului*

```
început temperaturi1
alocă T[7]
pentru zi = 1, 7 execută citește T[zi] sfârșit pentru
tmin ← T[1]
tmax ← T[1]
pentru zi = 2, 7 execută
    bloc
        dacă T[zi] < tmin atunci tmin ← T[zi]
        sfârșit dacă
        dacă T[zi] > tmax atunci tmax ← T[zi]
        sfârșit dacă
    sfârșit bloc
```



```
sfârșit pentru
scrie tmin
pentru zi = 1, 7
    dacă T[
    sfârșit d
sfârșit pentru
scrie tmax
pentru zi = 1, 7
    dacă T[
    sf dacă
sfârșit pentru
sfârșit temperatu
```

APLICAȚIA 2 FLOAR

Măsurarea temperaturii aerul
gești. Numărul voluntarilor dorn
clubului a organizat zece echipe
interes.

Prelucrarea valorilor înregist
– se afișează temperatura mi
pe parcursul săptămânii;
– se afișează temperatura me
– se afișează zona în care s-
pentru orice zi a săptămânii soli

1. *Analiza problemei*

Datele problemei au aceeași semnificație – temperatura zilnică –, dar se referă la zone diferite; de aceea, pentru organizarea lor se introduce și criteriul **zonă**. Rezultă un tablou (**T**) cu două dimensiuni: **zona** (linie) și **ziua** (coloană) (fig. 5).

Figura 5

```
        sfârșit pentru
            scrie tmin
            |pentru zi = 1, 7 execută
            |    |dacă T[zi] = tmin atunci scrie zi
            |    |sfârșit dacă
            |sfârșit pentru
            scrie tmax
            |pentru zi = 1, 7 execută
            |    |dacă T[zi] = tmax atunci scrie zi
            |    |sf dacă
            |sfârșit pentru
            sfârșit temperaturi1
```

APLICAȚIA 2 FLOARE DE COLȚ

Măsurarea temperaturii aerului în zonele greu accesibile a fost foarte apreciată de ecologiști. Numărul voluntarilor dornici să participe la astfel de acțiuni a crescut. Instructorul clubului a organizat zece echipe care să măsoare temperatura zilnică în diverse zone de interes.

Prelucrarea valorilor înregistrate se va face astfel:

– se afișează temperatura minimă și temperatura maximă înregistrate în cele zece zone pe parcursul săptămânii;

– se afișează temperatura medie, pentru oricare dintre zone, la solicitarea celui interesat;

– se afișează zona în care s-a atins cea mai mică, respectiv cea mai mare temperatură, pentru orice zi a săptămânii solicitată de cel interesat.

1. *Analiza problemei*

Datele problemei au aceeași semnificație – temperatura zilnică –, dar se referă la zone diferite; de aceea, pentru organizarea lor se introduce și criteriul **zonă**. Rezultă un tablou (**T**) cu două dimensini: **zona** (linie) și **ziua** (coloană) (fig. 5).

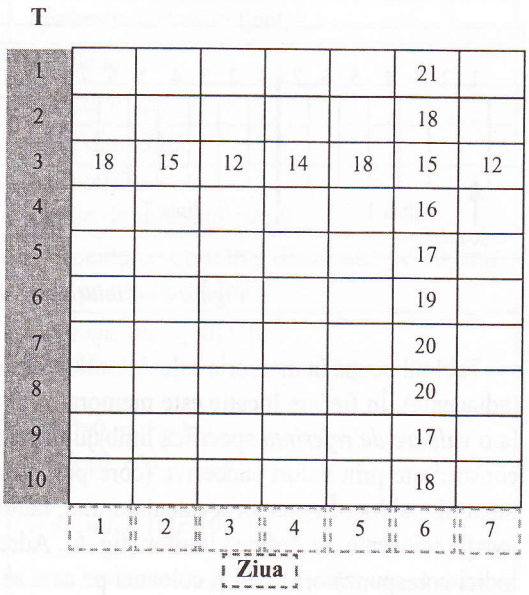


Figura 5

Cum organizăm datele în tablouri

1. Dacă într-o problemă este necesară memorarea mai multor valori cu aceeași semnificație, aceste valori vor fi grupate într-un ansamblu de tip **tablou**.
2. În funcție de cerințele problemei și de semnificația datelor, acestea sunt organizate în tablouri cu o singură dimensiune, numite **vectori**, sau în tablouri cu două dimensiuni, numite **matrice**.
3. Elementele unui tablou se identifică printr-o adresă formată din numele tabloului și câte un indice pentru fiecare dimensiune.
4. La tablourile cu două dimensiuni, primul indice din adresă reprezintă linia, iar cel de-al doilea coloana tabloului.
5. Operațiile care se repetă pentru fiecare element din tablou pot fi grupate în structuri repetitive cu contor. Contorul generează chiar indicele de adresă.

3. IMPLEMENTAREA TABLOURILOR BIDIMENSIONALE

Pentru memorarea și prelucrarea datelor organizate ca tablouri bidimensionale, înainte de scrierea unui program trebuie să cunoaștem următoarele elemente:
– tipul elementelor din tablou (datele cu aceeași semnificație),
– numărul maxim de elemente din tablou (capacitatea tabloului).

capacitatea tabloului = nrmax _linii * nrmax _coloane

Aceste elemente rezultă din analiza problemei și sunt folosite de compilator pentru determinarea zonei de memorie alocată tabloului.

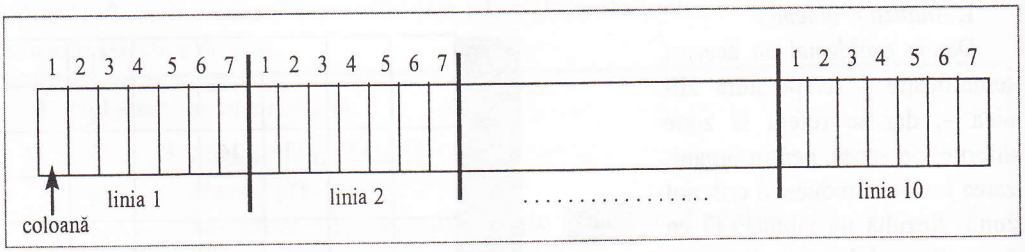


Figura 6 – Liniarizarea tabloului bidimensional

Tabloul ocupă în memoria calculatorului o „suprafață" (*array*) compusă din locații învecinate (adiacente). În fiecare locație este memorată valoarea unui element. Adresa locațiilor începe de la o *valoare de referință* specifică limbajului de programare (corespunzătoare primei linii) și se construiește prin valori succesive (corespunzătoare coloanelor de pe linie), pentru fiecare element din tablou. În memoria calculatorului, tabloul este liniarizat: elementele sunt memorate în locații adiacente, în ordinea liniilor (fig. 6). Adresarea unui element se face printr-o pereche de indici corespunzători liniei și coloanei pe care se află elementul respectiv.

În consecință, tipul variabilelor folosite ca indice de adresă trebuie să accepte valori în domeniile:

(1) [*valoare de referință, nrmax-linii*] pentru indicele de linie și

(2) [*valoare de referință, nrmax-coloane*] pentru indicele de coloană.

Adresarea elementelor se face, cel mai frecvent, prin două structuri repetitive cu contor, imbricate:

```
pentru  indice_linie = valoare-de referință la nr-linii execută
        pentru  indice_coloana = valoare-de referință la nr-coloane execută
          // prelucrarea elementelor din tablou în ordinea așezării pe linii
        sfârșit pentru
sfârșit pentru
```

Pentru indicele de linie sau indicele de coloană, se pot folosi tipuri diferite de date, ceea ce permite o referire asemănătoare cu cea întâlnită la jocul de șah; spre exemplu: T[2][C] reprezintă elementul de pe tabla de șah aflat pe linia 2 în coloana C (fig. 7).

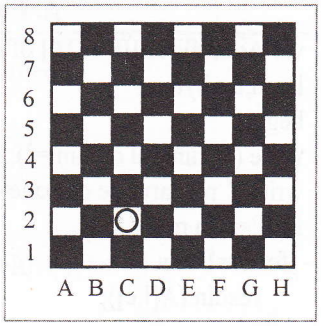


Figura 7

Elementele de sintaxă specifice tablourilor bidimensionale sunt prezentate în Tabelul 1.

***Tabelul* 1. Tablouri bidimensionale – elemente de sintaxă**

PASCAL	C / C++
Declararea tabloului	
var *tablou* **array** [1...*nl, 1..nc*] **of** *tipelement;*	*tipelement tablou*[*nl*][nc];
Declararea indicelui de adresă	
var *l,c:tipindice;*	*tipindice l,c;*
Adresarea unui element din tablou	
tablou [l,c] adresa de referință este 1.	*tablou* [*l*][*c*] adresa de referință este 0.
Exemplu: pentru tabloul *temperaturi* cu 100 de elemente de tip întreg distribuite pe 10 linii și 10 coloane: se inițializează cu zero elementele de pe primele 5 linii	

var *temperaturi* : **array** [1..10,1..10] **of integer**;
l,c: **byte;**
begin
for l:=1 to 5 do
for c:=1 to 10 do
temperaturi [l,c] =0;
end
	int *temperaturi* [10] [10];
	short l,c;
	{
	for (l=0; i< 5; i++)
	for (c=0; i< 10; i++)
	
	temperaturi [l][c]=0;
	}

Datele organizate în tablouri bidimensionale sunt prelucrate element cu element. Iată câteva dintre cele mai frecvente prelucrări:

- introducerea valorilor direct de la tastatură sau dintr-un fișier;
- afișarea valorilor pe ecran sau într-un fișier;
- verificarea unor proprietăți;
- compararea valorilor (aflarea elementului maxim sau minim);
- simularea unor situații reale.

***Tabelul 2.* Operații la nivel de matrice**

PASCAL	C / C++
A. Introducerea valorilor în ordinea „pe linii"	

var A: array[1..10, 1..10] of integer;
l,c,m,n : byte;
begin
write (' numarul de linii='); readln (m);
write (' numarul de coloane='); readln (n);
for l:=1 to m do
for c:=1 to n do
readln (A[l,c]);
end;
	int A [10] [10] ;
	int l,c,m,n ;
	{
	cout<<" numarul de linii=" ; cin>>m;
	cout<<" numarul de coloane=" ; cin>>n
	for (l=0; l< m; l++)
	for (c=0; c< n; c++)
	cin >>A[l][c];
	}

B. Afișarea valorilor în ordinea „pe coloane" se păstrează declarațiile de la secvența A	

begin
for c:=1 to n do
begin
writeln;
for l:=1 to m do
write (A[l,c], ' ');
end;
end;
	for (c=0; c< n; c++)
	{
	cout<<endl;
	for (l=0; l< m; l++)
	cout <<A[l][c]<< " ";
	}

C. Determinarea elementului minim de pe o linie oarecare, k se păstrează declarațiile de la secvența A	

var k:byte; min: integer;
begin
write ('specificati linia='); readln (k);
min := A[k,1];
for c:=2 to n do
if A[k,c] < min then
min:= A[k,c];
write ('min. de pe linia', k, '=', min);
end;
	int k, min;
	{
	cout<<" specificati linia=" ; cin>>k;
	min=A[k][0];
	for (c=1; c< n; c++)
	if A[k][c] < min
	min= A[k][c];
	cout<<"min. de pe linia"<<k<<"="<<min;
	}

nent cu element. Iată câteva

PASCAL	C / C++
D. Simularea unor situații reale.	
Exemplu: memorarea relațiilor de prietenie dintre membrii unui grup de n persoane (n<=10)	

var R : array [1..10, 1..10] of byte;
l,c,i,j,d,n : byte;
begin
write (' numarul de persoane=');
readln (n);
d:=1;
repeat
write ('prietenie intre: '); readln (i,j);
{relatia de pritenie este reciproca}
R[i,j] :=1; R[j,i] :=1;
write ('mai sunt prieteni? ');
readln (d);
{ da (d=1)/ nu (d=0)}
until d=0;
for l:=1 to n do
begin
writeln;
for c:=1 to n do
write (R[l,c], ' ');
end;
end;
	int R[10] [10] ;
	int l,c,i,j,d=1,n ;
	{
	cout<<" numarul de persoane=" ; cin>>n;
	while (d)
	{
	cout<<"prietenie intre:" ; cin>>i>>j;
	// relatia de prietenie este reciproca
	R[i][j]=1; R[j][i]=1;
	cout<<" mai sunt prieteni?" ; cin>>d;
	// da (d=1)/ nu (d=0)
	}
	for (l=0; l< n; l++)
	{
	cout<<endl;
	for (c=0; c< n; c++)
	cout>>R[l][c]<<" ";
	}
	}

Reluăm aplicația Floare de colț

Din analiza problemei, a reieșit necesitatea organizării datelor într-un tablou bidimensional. Continuăm rezolvarea problemei.

2. *Raționamentul problemei*

Pentru determinarea temperaturii minime și a temperaturii maxime înregistrate în cele zece zone, pe parcursul săptămânii, se prelucrează toate elementele din tablou.

Pentru determinarea temperaturii medii, **tmed**, se solicită zona și se prelucrează doar elementele de pe linia corespunzătoare.

Pentru determinarea zonei în care s-a atins temperatura minimă sau maximă, se solicită ziua și se prelucrează doar coloana corespunzătoare.

3. *Reprezentarea algoritmului*

```
început temperaturi2
alocă T[10, 7]
pentru zona = 1, 10 execută
    pentru zi = 1, 7 execută
        citește T[zona, zi]
    sfârșit pentru
sfârșit pentru
tmin ← T[1, 1]
tmax ← T[1, 1]
pentru zona = 1, 10 execută
    pentru zi = 1, 7 execută
        dacă T[zona, zi] < tmin
            atunci tmin ← T[zona, zi]
        sfârșit dacă
        dacă T[zona, zi] > tmax
            atunci tmax ← T[zona, zi]
        sfârșit dacă
    sfârșit pentru
sfârșit pentru
scrie tmin
scrie tmax
scrie "ce zonă vă interesează?"
citește zona
tmed ← 0
pentru zi = 1, 7 execută
    tmed ← tmed + T[zona, zi]
sfârșit pentru
tmed ← tmed/7
scrie zona, tmed
scrie "ce zi vă interesează?"
citește zi
tmin ← T[1, zi]
zmin ← 1
tmax ← T[1, zi]
zmax ← 1
pentru zona = 2, 10 execută
    bloc
        dacă T[zona, zi] < tmin
            atunci
                bloc
                    tmin ← T[zona, zi]
                    zmin ← zona
                sfârșit bloc
        sfârșit dacă
        dacă T[zona, zi] > tmax
            atunci
                bloc
                    tmax ← T[zona, zi]
                    zmax ← zona
                sfârșit bloc
        sfârșit dacă
    sfârșit bloc
sfârșit pentru
scrie zmin, tmin, zi
scrie zmax, tmax, zi
sfârșit temperaturi2
```

TEME

1. La cabinetul medical, se calculează înălțimea medie a tuturor băieților din școală. Precizați care este forma de organizare a datelor corespunzătoare acestei situații.
2. La cabinetul medical, se calculează înălțimea medie a tuturor băieților din școală, pe grupe de vârstă (ani de studiu). Precizați care este forma de organizare a datelor corespunzătoare acestei situații.
3. La cabinetul medical, se calculează înălțimea medie a tuturor băieților din școală pe grupe de vârstă (ani de studiu), pentru fiecare clasă în parte (9A,..., 12 C,...). Precizați care este forma de organizare a datelor corespunzătoare acestei situații.
4. Administratorul unui bloc cu 12 etaje și 20 de apartamente pe etaj calculează cheltuielile de întreținere, în funcție de numărul de persoane din fiecare apartament (toate apartamentele au același număr de camere).

ed
interesează?"

i]

zi]

2, 10 execută

T[zona, zi] < tmin
tunci
bloc
tmin ← T[zona, zi]
zmin ← zona
sfârșit bloc
it dacă
T[zona, zi] > tmax
tunci
bloc
tmax ← T[zona, zi]
zmax ← zona
sfârșit bloc
it dacă
c

, zi
x, zi
uri2

băieților din școală. Precizați situații.

eților din școală, pe grupe de datelor corespunzătoare aces-

ieților din școală pe grupe de ...). Precizați care este forma

taj calculează cheltuielile de nent (toate apartamentele au

Realizați un program care să răspundă următoarelor cerințe:

a) înregistrarea numărului de persoane din fiecare apartament, pe etaje, de la parter până la ultimul etaj;

b) afișarea numărului de persoane din fiecare apartament, pe etaje, de la ultimul etaj până la parter;

c) cunoscând numărul unui apartament, introdus de la tastatură, să se afișeze etajul la care se află apartamentul și numărul de persoane care locuiesc în apartamentul respectiv.

5. Se consideră tabloul **M** cu următoarele elemente:

1	2	3	4
5	6	7	8
9	10	11	12

Precizați ce valori afișează secvența de mai jos:

```
pentru c = 1, 3 execută
    pentru l = 1, 2 execută
        scrie M[l, c]
    sfârșit pentru
sfârșit pentru
```

a) 1, 5, 9, 2, 6, 10, 3, 7, 11; b) 1, 2, 3, 5, 6, 7; c) 1, 5, 2, 6, 3, 7.

6. Ce realizează următoarea secvență de operații:

```
alocă M[10, 10]
pentru i = 1, 10 execută
    M[i, i] ← i
sfârșit pentru
```

a) atribuie valori de la 1 la 10 elementelor din vectorul **M**;

b) atribuie fiecărui element de pe diagonala matricei **M** o valoare egală cu linia pe care acesta se află;

c) atribuie valori de la 1 la 10 primelor 10 elemente din matricea **M**.

7. Care dintre următoarele secvențe de operații memorează în colțurile matricei **M** valori citite de la tastatură; matricea are 5 linii și 5 coloane.

a)
```
pentru i = 1, N execută
    pentru j = 1, N execută
        citește M[i, j]
    sfârșit pentru
sfârșit pentru
```

b)
```
pentru i = 1, 5 execută
    pentru j = 1, 5 execută
        dacă (i = 1) și (j = 5)
            atunci citește M[i, i]
            altfel citește M[j, j]
        sfârșit dacă
    sfârșit pentru
sfârșit pentru
```

c)
```
l ← 1
c ← 5
pentru x = 1, 4 execută
    citește M[l, c]
    c ← l
    l ← c
sfârșit pentru
```

d)
```
l ← 1
c ← 5
citește M[l, l], M[l, c], M[c, l], M[c, c]
```

4. TABLOURI BIDIMENSIONALE – CAZURI PARTICULARE

Matrice pătrată

Rezolvarea problemelor cu calculatorul necesită găsirea soluțiilor de organizare și memorare a datelor, astfel încât acestea să păstreze semnificațiile reale atât din punct de vedere al valorilor proprii, cât și al relațiilor cu alte date. Spre exemplu, dacă membrii unui grup (persoane) împrumută bani unii de la alții, interesează atât suma de bani primită/datorată, cât și cine/de la cine a împrumutat. În acest caz, dacă în grup sunt *n* persoane, un tablou bidimensional, *G*, cu *n* linii și *n* coloane, este suficient pentru păstrarea atât a valorilor împrumutate, cât și a relațiilor de împrumut (fig. 8).

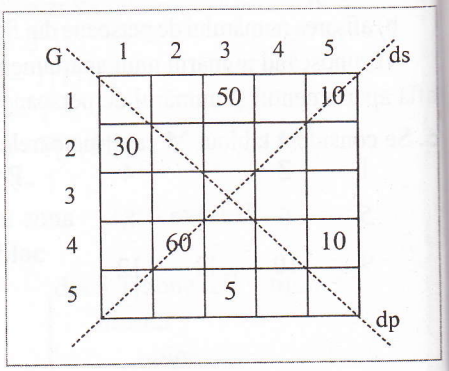


Figura 8

Fiecare element din tablou reprezintă valoarea unui împrumut; liniile și coloanele reprezintă persoanele din grup care dau bani unei alte persoane sau primesc bani de la altă persoană din grup. Fie *i* și *j* două persoane: *G[i,j]* reprezintă suma de bani pe care i-a împrumutat-o lui *j*, adică suma de bani pe care *j* a primit-o de la *i*. Pentru exemplul din figura 8, *G[4,2]* reprezintă suma de 60 lei pe care persoana 4 a împrumutat-o persoanei 2.

Tabloul folosit are o particularitate: numărul de linii este egal cu numărul de coloane, de aceea este numit tablou pătrat sau, mai simplu, *matrice pătrată.*

Într-o matrice pătrată deosebim următoarele elemente specifice (fig. 8):

– diagonala principală (*dp*);
– diagonala secundară (*ds*);
– triunghiurile formate de cele două diagonale;
– direcțiile paralele cu fiecare dintre cele două diagonale.

Matrice binară

Reluăm situația grupului de persoane care împrumută bani unii de la alții, dar urmărim numai relația de împrumut: cine/de la cine a primit bani. În acest caz, nu se mai păstrează valoarea împrumutului. Fie *i* și *j* două persoane: *G[i,j]* are semnificația *i* a împrumutat bani lui *j* echivalent cu *j* a primit bani de la *i* (fig. 9). Elementele matricei nu pot avea decât două valori alese convențional (spre exemplu 0 sau 1), cu semnificația:

G	1	2	3	4	5
1	0	0	1	0	1
2	1	0	0	0	0
3	0	0	0	0	0
4	0	1	0	0	1
5	0	0	1	0	0

Figura 9

$$G[i,j] = \begin{cases} 1 & \text{dacă } i \text{ împrumută bani lui } j \\ 0 & \text{dacă } i \text{ nu împrumută bani lui } j \end{cases}$$

ULARE

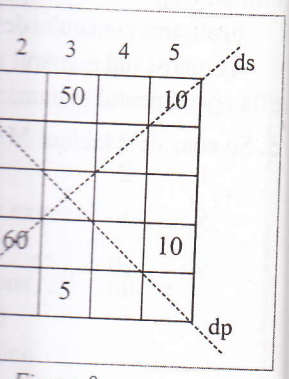


Figura 8

ile și coloanele reprezintă
ni de la altă persoană din
împrumutat-o lui *j*, adică
G[4,2] reprezintă suma

numărul de coloane, de

8):

	3	4	5
	1	0	1
	0	0	0
	0	0	0
	0	0	1
	1	0	0

gura 9

Matricea ale cărei elemente au valori în mulțimea {0,1}, cu semnificații logice complementare, se numește *matrice binară.*

Matrice simetrică

Dacă în grupul de *n* persoane urmărim relațiile de prietenie, acestea pot fi memorate tot într-o matrice binară ale cărei elemente au următoarea semnificație:

$$P[i,j] = \begin{cases} 1 \text{ dacă } i \text{ este prieten cu } j \\ 0 \text{ dacă } i \text{ nu este prieten cu } j \end{cases}$$

Prietenia este o relație reciprocă; acest aspect se regăsește în proprietatea de *simetrie* a matricei binare asociată grupului de persoane: *P [i, j] = P [j, i]* (fig. 10).

P	1	2	3	4	5
1			1	1	
2			1		1
3	1	1		1	
4	1		1		1
5		1		1	

Figura 10

Matricea punctelor unui plan

Pe ecranul monitorului, în modul de lucru text, caracterele sunt afișate pe rânduri, de sus în jos, de la stânga la dreapta, pe fiecare rând. Se poate spune că ecranul monitorului este o suprafață plană ale cărei puncte sunt distribuite pe linii și coloane (cel mai frecvent, 24 de linii și 80 de coloane). În acest exemplu, regăsim modelul bidimensional de organizare a datelor: oricărei suprafețe plane îi poate fi asociată o *matrice a punctelor*. Valorile atribuite elementelor din matrice au semnificația specifică problemei modelate. Pentru exemplul suprafeței-ecran, dacă elementele matricei sunt de tip caracter, în matrice poate fi reținut textul de pe un ecran.

Un alt exemplu: o imagine – fotografie – este tot o reprezentare în plan. Punctele planului formează obiecte distincte, dacă sunt evidențiate diferit de la un obiect la altul, prin culoare. Dacă toate punctele unei imagini (suprafață) sunt colorate la fel, imaginea este formată dintr-un singur obiect.

Oricărei imagini îi poate fi asociată o matrice a punctelor; valorile atribuite elementelor din matrice au semnificația culorii fiecărui punct. În figura 11 este reprezentată matricea asociată unei imagini, în care s-au folosit 3 coduri de culori cu semnificația: 0 – alb, 1– negru, 2 – roșu.

Cu ajutorul matricelor de tip plan, pot fi modelate și situații de joc: așezarea pieselor pe tabla de șah, configurația unui labirint, „X și O", „avioane" și altele.

0	0	0	1	0	0	0	0
0	1	1	1	1	1	0	0
1	2	2	2	2	0	0	0
1	2	2	2	1	0	0	0
1	2	2	1	2	0	0	0
0	1	1	2	1	1	1	0
0	0	1	2	1	2	1	0
0	1	1	1	0	1	1	0

Figura 11

TEME

1. Determinați proprietățile elementelor aflate pe diagonala principală, într-o matrice pătrată. Scrieți un program pentru afișarea acestor elemente.
2. Determinați proprietățile elementelor aflate pe diagonala secundară, într-o matrice pătrată. Scrieți un program pentru afișarea acestor elemente.
3. Scrieți un program care să verifice dacă o matrice pătrată este simetrică.
4. Formulați un exemplu de problemă care să necesite organizarea datelor într-o matrice binară.
5. Formulați un exemplu de problemă care să necesite organizarea datelor într-o matrice de tip plan.
6. Determinați condițiile de amplasare pe tabla de șah a două piese de joc – dame –, astfel încât acestea să nu se atace. (Indicație: două piese de șah – dame – se atacă dacă sunt amplasate pe aceeași linie, pe aceeași coloană sau pe aceeași diagonală.)
7. Construiți tabloul de vecinătate pentru țările situate pe harta din figura 12.

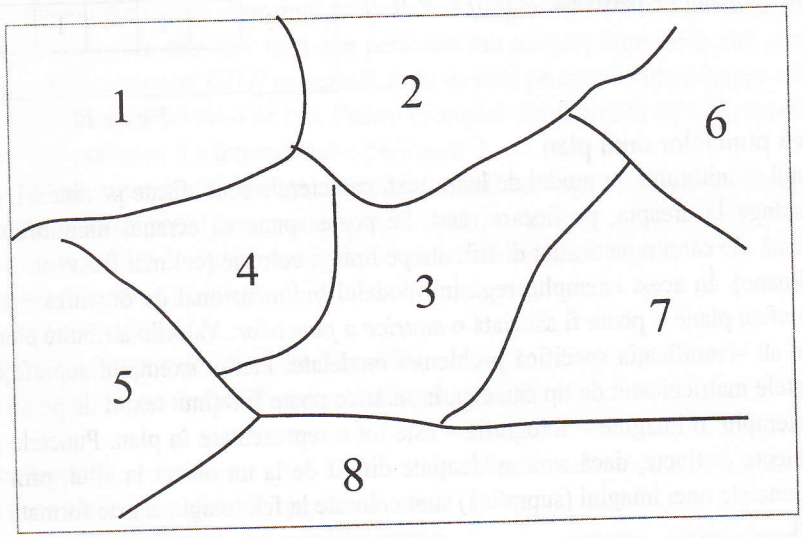


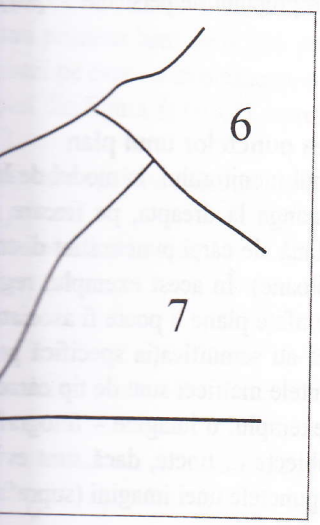
Figura 12

8. Să se verifice dacă o matrice pătrată este „tablou magic". Într-un tablou magic, suma elementelor de pe oricare linie este egală cu suma elementelor de pe oricare dintre coloane precum și cu suma elementelor de pe oricare dintre diagonale.

Exemplu:

3	2	7
8	4	0
1	6	5

gonala principală, într-o matrice pătrată.
te.
agonala secundară, într-o matrice pătrată.
e.
pătrată este simetrică.
e organizarea datelor într-o matrice binară.
te organizarea datelor într-o matrice de tip
a două piese de joc – dame –, astfel încât
ah – dame – se atacă dacă sunt amplasate
diagonală.)
pe harta din figura 12.



magic". Într-un tablou magic, suma ele-
nentelor de pe oricare dintre coloane pre-
agonale.

5. PRELUCRAREA TABLOURILOR BIDIMENSIONALE

5.1. Localizarea elementelor cu aceeași proprietate

Formațiuni geografice

Se dorește determinarea configurației unui teren dreptunghiular după formațiunile geografice din Tabelul 3 și figura 13.

Analiza terenului se face prin secționarea acestuia pe orizontală și pe verticală.

Punctele aflate la intersecția dintre secțiunile orizontale și secțiunile verticale sunt cotate față de nivelul mării; cotele sunt valori numerice întregi și pozitive.

*Terenul este secționat prin **n** secțiuni orizontale și **m** secțiuni verticale.*

O formațiune geografică este formată din cel puțin trei puncte.

***Tabelul 3.* Formațiuni geografice**

FORMAȚIUNEA GEOGRAFICĂ	
Pantă	a)
Râpă	b)
Deal	c)
Vale	d)
Platou	e)
Teren denivelat	f)
Punct de tip *șa_xy*: punct aflat la cota maximă pe secțiunea orizontală *x* și la cota minimă pe secțiunea verticală *y*; se poate defini și punct de tip *șa_yx*	g)

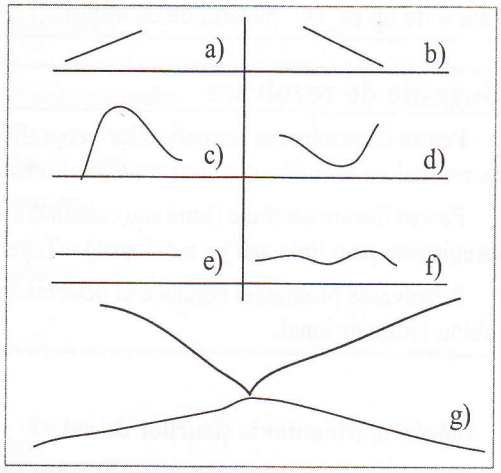


Figura 13

Exemplu:

În Tabelul 4 este reprezentat un teren pe care s-au făcut patru secțiuni orizontale și șapte secțiuni verticale.

***Tabelul 4.* Înregistrarea datelor într-un teren secționat**

	1	2	3	4	5	6	7
1	25	72	69	69	69	73	40
2	20	40	50	56	30	19	100
3	15	30	35	40	39	20	10
4	10	55	0	60	42	50	19

Rezultatele analizei pe secțiuni sunt prezentate în tabelul Tabelul 5.

Tabelul 5. **Analiza pe secțiuni**

Secțiuni orizontale		Secțiuni verticale	
Numărul secțiunii	**Formațiunea geografică**	**Numărul secțiunii**	**Formațiunea geografică**
1	Platou la cota 69	1	Râpă
2	Deal cu vârf la cota 56	2	Vale cu punct minim la cota 30
3	Deal cu vârf la cota 40	3	Râpă
4	Teren denivelat	4	Vale cu punct minim la cota 40
		5	Teren denivelat
		6	Vale cu punct minim la cota 19
		7	Teren denivelat
Puncte de tip șa_xy: punctul de coordonate [3,4] la cota 40			

Sugestie de rezolvare

Pentru determinarea formațiunilor geografice, cotele vor fi înregistrate într-un tablou bidimensional cu semnificația: linii – secțiuni orizontale, coloane – secțiuni verticale.

Pentru fiecare secțiune (linie sau coloană) se va studia monotonia șirurilor de valori (cotele înregistrate pe o linie sau pe o coloană) – Tabelul 6.

Rezolvarea problemei conduce la determinarea elementelor cu aceeași proprietate dintr-un tablou bidimensional.

Tabelul 6. **Monotonia șirurilor de valori**

Formațiunea geografică	Monotonia șirului de valori
Pantă/ Râpă	Șir crescător/ descrescător
Deal	Există un punct de tip vârf, astfel încât toate punctele dispuse la stânga acestuia formează un șir monoton crescător, iar toate punctele dispuse la dreapta acestuia formează un șir monoton descrescător.
Vale	Există un punct de cota minimă, astfel încât toate punctele dispuse la stânga acestuia formează un șir monoton descrescător, iar toate punctele dispuse la dreapta acestuia formează un șir monoton crescător.
Platou	Există cel puțin trei puncte consecutive aflate la aceeași cotă.

belul Tabelul 5.

Secțiuni verticale	
rul nii	Formațiunea geografică
	Râpă
	Vale cu punct minim la cota 30
	Râpă
	Vale cu punct minim la cota 40
	Teren denivelat
	Vale cu punct minim la cota 19
	Teren denivelat
a 40	

vor fi înregistrate într-un tablou bidi-
loane – secțiuni verticale.
a monotonia șirurilor de valori (cotele
entelor cu aceeași proprietate dintr-un

Monotonia șirului de valori
ător/ descrescător
punct de tip vârf, astfel încât toate dispuse la stânga acestuia formează onoton crescător, iar toate punctele la dreapta acestuia formează un șir descrescător.
punct de cota minimă, astfel încât nctele dispuse la stânga acestuia un șir monoton descrescător, iar nctele dispuse la dreapta acestuia un șir monoton crescător.
puțin trei puncte consecutive aflate cotă.

Pentru exemplificarea rezolvării, prezentăm, în pseudocod, secvența prelucrărilor pentru determinarea formațiunii de tip *platou* și a punctelor de tip *șa_xy* .

Determinarea formațiunii de tip platou

```
început platou
// căutarea unei formațiuni platou pe linia k
//semnificația variabilelor de lucru
//M[100,100] tabloul cotelor cu m secțiuni verticale
// lp lungimea platoului
// cp cota platoului
// inițalizare lungime platou
lp←1
cp←M[k,1]
pentru c=2 la m execută
// se verifică dacă punctul M[k,c] aparține platoului
   dacă M[k,c] =cp
         atunci lp ← lp+1  // crește lungimea platoului
         altfel
            dacă lp>=3    // există platou la copta cp
               atunci scrie platou la cota cp
            sfârșit dacă
// inițializări pentru determinarea unui nou platou
         lp←1
         cp←M [k,c]
   sfârșit dacă
sfârșit pentru
// se verifică dacă linia k se termină cu platou
dacă lp>=3
   atunci scrie platou la cota cp
sfârșit dacă
sfârșit platou
```

Determinarea punctelor de tip șa_xy

- Raționamentul de rezolvare

Pasul 1

Se parcurge tabloul pe linii: pentru fiecare linie, se determină elementul maxim și se păstrează coloana acestuia în vectorul *max_linii*.

Pentru exemplul dat, vectorul *max_linii* are următoarele valori: 6, 7, 4, 4.

Pasul 2

Se parcurge tabloul pe coloane; pentru fiecare coloană, se determină elementul minim și se păstrează linia acestuia în vectorul *min_coloane*.

Pentru exemplul dat, vectorul *min_coloane* are următoarele valori: 4, 3, 4, 3, 2, 2, 3.

Pasul 3

Se parcurge vectorul *max_linii*: în variabila de lucru *cmax,* se reține valoarea unui element *max_linii[k]*

$$cmax = max\_linii[k]$$

se verifică proprietatea de punct șa_xy:

$$min\_colane[cmax] = k$$

Pentru exemplul dat, urmărim datele din Tabelul 7.

Există punct șa_xy pe linia 3, coloana 4.

Tabelul 7. **Determinarea punctului șa_xy**

linia	Cmax	min-coloane[cmax]
1	6	2
2	7	3
3	4	3
4	4	3

- Secvența pseudocod a prelucrărilor pentru determinarea punctelor de tip *șa_xy*:

```
început  punct_șa_xy
// căutarea unui punct șa_xy
//semnificația variabilelor de lucru
//M[100,100]  tabloul cotelor cu  m secțiuni orizontale  și n secțiuni verticale
// max_linii [100]  vector  cu m elemente în care se reține coloana  elementului maxim de pe
        fiecare linie
// min_coloane[100]  vector  cu n elemente în care se reține linia  elementului minim de pe
        fiecare coloană
//max    valoarea maximă   pe o linie;  cm coloana pe care se află  max
//min    valoarea minimă   pe o coloană;  lm linia  pe care se află  min

// se determină elementul maxim de pe fiecare linie
pentru k=1 la m execută
  max ← M[k,1]
  cm ← 1
  pentru c=2  la  n execută
   dacă  M[k,c] >max
            atunci
                bloc
                max← M[k,c]
                cm ← c
               sfârșit bloc
    sfârșit dacă
  sfârșit pentru
```



```
// în linia k, elementul maxim se
max_linii[k] ←  cm
sfârșit  pentru

// se determină elementul minim
pentru c=1 la n execută
  min ← M[1,c]
  lm ← 1
  pentru k=2 la m execută
   dacă  M[k,c] <min
            atunci
                bloc
                min ← M[k,c]
                lm ← k
               sfârșit bloc
    sfârșit dacă
  sfârșit pentru
// în coloana c,  elementul minim
min_coloane[c] ← lm
sfârșit pentru
// se parcurge vectorul max_linii
pentru k=1 la m execută
 cmax ← max_linii [k]
// se verifică proprietatea de punct
 dacă min_coloane[cmax]=k
            atunci
                scrie punct
  sfârșit dacă
sfârșit pentru
sfârșit punct _sa_xy
```

TEME

1. Construiți exemple numerice pentru
   a) platou pe coloana 3,
   b) deal pentru linia 1,
2. Construiți expresii pentru relații geografice.
3. Codificați, în limbajul de programare, mulțimii geografice de tip platou.
4. Codificați, în limbajul de programare, punctelor de tip șa_xy.

se reține valoarea unui element

ța_xy

[cmax]

unctelor de tip *șa_xy*:

secțiuni verticale
oana elementului maxim de pe
linia elementului minim de pe
e află **max**
află **min**

```
// în linia k, elementul maxim se află pe coloana cm
max_linii[k] ← cm
sfârșit pentru

// se determină elementul minim de pe fiecare coloană
pentru c=1 la n execută
  min ← M[1,c]
  lm ← 1
  pentru k=2 la m execută
   dacă M[k,c] <min
            atunci
                 bloc
                 min ← M[k,c]
                 lm ← k
                sfârșit bloc
    sfârșit dacă
  sfârșit pentru
// în coloana c, elementul minim se află pe linia lm
min_coloane[c] ← lm
sfârșit pentru
// se parcurge vectorul max_linii
pentru k=1 la m execută
 cmax ← max_linii [k]
// se verifică proprietatea de punct șa_xy
 dacă min_coloane[cmax]=k
            atunci
                    scrie punct sa de coordonate k, cmax
 sfârșit dacă
sfârșit pentru
sfârșit punct _sa_xy
```

TEME

1. Construiți exemple numerice pentru următoarele formațiuni geografice:
   a) platou pe coloana 3, c) râpă pentru coloana 5, e) punct șa_xy,
   b) deal pentru linia 1, d) vale pentru linia 4, f) punct șa_yx.
2. Construiți expresii pentru relațiile de monotonie corespunzătoare fiecărei formațiuni geografice.
3. Codificați, în limbajul de programare studiat, secvența pseudocod pentru determinarea formațiunii geografice de tip platou.
4. Codificați, în limbajul de programare studiat, secvența pseudocod pentru determinarea punctelor de tip șa_xy.

5. Realizați, în limbajul de programare studiat, un program pentru determinarea punctelor d tip șa_yx.

6. Scrieți secvențele de instrucțiuni pentru determinarea următoarelor formațiuni geografice: a) râpă, b) deal, c) vale.

7. Realizați și testați programul pentru derminarea configurației unui teren ale cărui coordonat și cote se citesc din fișierul *teren.in* cu următoarea structură:

– pe prima linie valorile *n* și *m* reprezentând: *n* numărul de secțiuni orizontale și *m* număru de secțiuni verticale;

– pe următoarele *n* linii câte *m* valori reprezentând cotele aflate pe fiecare secțiune orizontală

5.2. Prelucrarea elementelor distribuite pe aceleași direcții (linii, coloane, diagonale)

Aranjamente florale

Un grădinar are mai multe soiuri de plante pe care dorește să le planteze atât în aranjamente clasice, cât și în forme noi; spre exemplu, în locul rondurilor, grădinarul vrea să compună careuri florale. Întrucât timpul de creștere și înflorire al plantelor nu poate fi întârziat, grădinarul și-a propus să testeze modelele cu calculatorul.

Analiza problemei

Pentru rezolvarea cu calculatorul, soiurile de plante decorative vor fi codificate. În Tabelul 8 se prezintă un exemplu de codificare.

Tabelul 8. **Codificarea plantelor decorative**

Planta decorativă/culoare	Codul plantei
Iarbă/verde	1
Trandafir imperial/roșu	2
Narcise/albe	6
Narcise/galbene	7
Crizanteme/mov	8
Crizanteme/albe	9
Tuia/verde	11

Careul pe care va fi probat modelul este format din *n*m* sau *n*n* puncte; în fiecare punc poate fi sădită o plantă. În Tabelul 9 sunt prezentate câteva modele după care se vor testa aran jamentele florale.

Tabelul 9. Modele pentru

Denumirea modelului
Brazde paralele orizontale
Triunghi
Diagonale paralele cu diagon principală
Spirală

Sugestie de rezolvare

Careurile florale vor fi ge liniii și o coloane, în funcție d vor fi sădite plantele și se în

n pentru determinarea punctelor de

rmătoarelor formațiuni geografice:
vale.

rației unui teren ale cărui coordonate
tură:
l de secțiuni orizontale și *m* numărul

aflate pe fiecare secțiune orizontală

e aceleași direcții

*rește să le planteze atât în aranja-
ondurilor, grădinarul vrea să com-
e al plantelor nu poate fi întârziat,*

rative vor fi codificate. În Tabelul

decorative

odul plantei
1
2
6
7
8
9
11

sau $n*n$ puncte; în fiecare punct
nodele după care se vor testa aran-

Tabelul 9. **Modele pentru aranjamente florale**

Denumirea modelului	Exemplu de model
Brazde paralele orizontale	Decor de primăvară cu narcise albe și galbene: 6 6 6 6 6 7 7 7 7 7 6 6 6 6 6
Triunghi	Decor de primăvară cu narcise albe și galbene; aleea centrală (diagonală) cu gazon: 1 7 7 7 7 6 1 7 7 7 6 6 1 7 7 6 6 6 1 7 6 6 6 6 1
Diagonale paralele cu diagonala principală	Decor cu trandafiri imperiali și tuia plantați pe direcții paralele cu aleea centrală (diagonală): 1 2 11 2 11 2 1 2 11 2 11 2 1 2 11 2 11 2 1 2 11 2 11 2 1
Spirală	Decor de toamnă cu crizanteme mov și albe: 8 8 8 8 8 8 8 9 9 9 9 8 8 9 8 8 9 8 8 9 8 8 9 8 8 9 9 9 9 8 8 8 8 8 8 8

Sugestie de rezolvare

Careurile florale vor fi generate într-un tablou bidimensional, G, cu *n* linii și *m* coloane, sau *n* linii și *n* coloane, în funcție de model. Pentru fiecare model, se determină adresele punctelor în care vor fi sădite plantele și se înregistrează la aceste adrese codul plantei corespunzător modelului.

Rezolvarea problemei conduce la umplerea matricei cu valori dispuse pe direcții specifice modelului: linii, coloane, triunghiuri, diagonale, spirală.

În Tabelul 10 sunt prezentate secvențele pseudocod pentru generarea adreselor de umplere corespunzătoare modelelor prezentate.

Tabelul 10. **Secvențe pseudocod pentru generarea adreselor de umplere**

Modelul floral	Direcțiile de umplere	Generarea adreselor

Brazde orizontale	Linii	// brazda de tip linie
		// linia i
		pentru c=1 la m **execută**
		G[i,c] ← cod
		sfârșit pentru

Triunghi	Triunghi stânga	//se lucrează cu o matrice pătrată **n*n**
		// pe linii **i** și coloane **j**
		pentru i=2 la n **execută**
		pentru j=1 la i-1 **execută**
		G[i,j] ← cod
		sfârșit pentru
		sfârșit pentru

Paralele la diagonala principală	Se pot forma n-2 direcții **(d)** paralele cu diagonala principală și situate deasupra acesteia	//se lucrează cu o matrice **n*n**
		// pe linii **i** și coloane **j**
		// pe diagonalele **d**
		
		pentru d=1 la n-2 **execută**
		pentru i=1 la n-d **execută**
		G[i,i+d] ← cod
		sfârșit pentru
		sfârșit pentru

Spirală	Se parcurg cadranele de la exterior – primul cadran-**p**, spre interior, ultimul cadran-**u**	u p←1 u←n
		repetă
		// se parcurge prima linie, **p**, din cadran,
		//de la stânga la dreapta
		pentru j=p **la** u **execută**
		G[p,j] ← cod
		sfârșit pentru
		// se parcurge ultima coloană, **u**, din cadran,
		// de sus în jos
		pentru i=p+1 **la** u **execută**
		G[i,u] ← cod
		sfârșit pentru
		//se parcurge ultima linie, **u**, din cadran

valori dispuse pe direcții specifice

tru generarea adreselor de umplere

reselor de umplere

enerarea adreselor

inie

execută

o matrice pătrată **n*n**
loane **j**
execută
-1 **execută**

o matrice **n*n**
loane **j**
d

-2 **execută**
la n-d **execută**
,i+d] ← cod
ru

ima linie, **p**, din cadran,
dreapta
execută

tima coloană, **u**, din cadran,

u execută

ima linie, **u**, din cadran

		//de la dreapta la stânga
		pentru j=u -1 **la** p **execută**
		G[u,j] ← cod
		sfârșit pentru
		// se parcurge prima coloană, **p**, din cadran,
		// de jos în sus
		pentru i=u - 1 **la** p - 1 **execută**
		G[i,p] ← cod
		sfârșit pentru
		
		// se pregătesc valorile **p** și **u**
		// pentru cadranul următor
		p← p+1
		u←u-1
		// se testează dacă se mai pot forma cadrane
		până când p>u

TEME

1. Construiți expresii pentru relațiile care definesc direcțiile de umplere pentru fiecare dintre modelele propuse.
2. Scrieți secvențele de instrucțiuni pentru generarea fiecărui model.
3. Realizați și testați programul pentru generarea fiecărui model floral (pentru fiecare model se va scrie un program).
4. Pentru atractivitatea prezentării, realizați un program care să afișeze modelul floral colorat, corespunzător culorii de cod. (Indicație: modul de lucru text permite setarea atributului de culoare atât pentru fond, cât și pentru text.)
5. Realizați un program care să permită utilizatorului (grădinarul) să aleagă, pe rând, oricare dintre modelele oferite (program cu meniu – fig. 14).

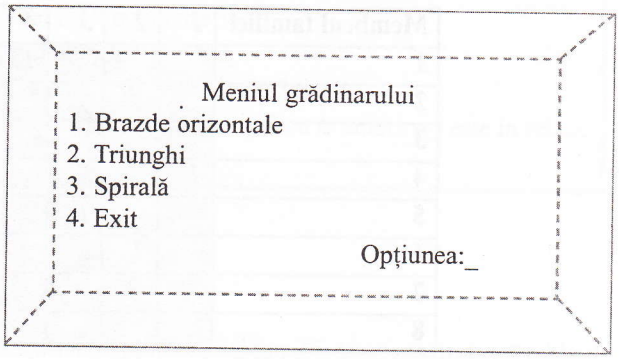


Figura 14

6. Rescrieți secvența umplerii în spirală astfel încât să folosiți cât mai puține structuri repetitive.
7. Compuneți un model floral nou și scrieți secvența de instrucțuni (programul) pentru generarea modelului propus.

8. Analizați următoarele secvențe pseudocod și determinați modelul de umplere generat:

a)	b)

pentru i=2 **la** n **execută**
  pentru j=n, **la** n+2-i **execută**
  G[i,j] ← cod
  sfârșit pentru
sfârșit pentru
	pentru i= 1 **la** [n/2] **execută**
	  pentru j=i+1 **la** n-i **execută**
	  G[i,j] ← cod
	  sfârșit pentru
	sfârșit pentru

c)	d)

pentru i=n **la** [n/2]+1 **execută**
  pentru j=n **la** n+2-i **execută**
  G[i,j] ← cod
  sfârșit pentru
sfârșit pentru
	pentru i=1 **la** n **execută**
	  pentru j=1, la **m** execută
	  G[i,j] ← cod
	  sfârșit pentru
	sfârșit pentru

5.3. Simularea unor situații reale

Tabloul de familie

Se consideră o familie formată din părinți (1) și copii (2). Fiecare copil are doi părinți între care există relație de căsătorie. Nu toți membrii familiei sunt căsătoriți; nu toți membrii familiei au copii.

Relațiile dintre membrii familiei sunt păstrate în tabloul de familie (fig. 15).

Figura 15. Tablou pentru o familie formată din 8 persoane

Membrul familiei	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1				*1*		*2*		*2*
2					*2*	*1*	*2*	
3					*1*			
4	*1*					*2*		*2*
5			*1*					
6		*1*			*2*		*2*	
7								
8								

odelul de umplere generat:

xecută execută
tă ecută

ecare copil are doi părinți între
ătoriți; nu toți membrii familiei

familie (fig. 15).

8 persoane

6	7	8
2		2
1	2	
2		2
	2	

Interpretarea tabloului de familie este prezentată în Tabelul 11.

Tabelul 11.

Membrul familiei	Căsătorit cu	Copii	Părinți
1	4	6 și 8	
2	6	5 și 7	
3	5		
4	1	6 și 8	
5	3		2 și 6
6	2	5 și 7	1 și 4
7			2 și 6
8			1 și 4

După cum rezultă din Tabelul 11, fiecare membru al familiei este complet caracterizat prin analiza înregistrărilor de tip linie și a înregistrărilor de tip coloană.

Analiza înregistrărilor de tip linie:

• familia are 8 membri; pentru fiecare membru (i) analizăm valorile de pe linia i din tablou, cu semnificația: (1) – i este căsătorit cu j, unde j este poziția valorii 1 pe linie,
(2) – i este părinte pentru k, unde k este poziția valorii 2 pe linie.

Analiza înregistrărilor de tip coloană:

• familia are 8 membri; pentru fiecare membru (j) analizăm valorile de pe coloana j din tablou cu semnificația: (1) – j este căsătorit cu i, unde i este poziția valorii 1 pe coloană,
(2) – k este părinte pentru j, unde k este poziția valorii 2 pe coloană.

Concluzii și restricții

- oricărei familii îi poate fi asociat un tablou de familie de tip matrice pătrată;
- pe o linie poate fi înregistrată cel mult o valoare 1;
- pe o coloană poate fi înregistrată cel mult o valoare 1;
- pe o coloană pot fi înregistrate cel mult două valori 2;
- dacă i este în relație de tip 1 cu j, și j este în relație de tip 1 cu i;
- dacă i este în relație de tip 1 cu j, și i este în relație de tip 2 cu k, atunci și j este în relație de tip 2 cu k.

TEME

1. Cunoscând relațiile de tip 1 sau 2 dintre cei n membri ai unei familii, să se construiască tabloul de familie.
2. Validarea tabloului de familie: fiind dat un tablou de familie, să se verifice corectitudinea înregistrărilor.

3. Fiind dat un tablou de familie validat, caracterizați fiecare membru al familiei.

4. Să se determine descendenții unui membru al familiei (pentru exemplul analizat, 1 are copii pe 6 și 8 și nepoți pe 5 și 7).

5. **a)** Realizați tabloul de familie pentru familia cu 10 membri din exemplul următor:

Membrul familiei	Căsătorit cu	Copii
2	3	1, 5, 7
5	8	1, 10
10	4	9

b) Analizați tabloul de familie obținut la cerința **a)** și precizați ce restricții au fost încălcate.

6. Cum procedăm pentru a determina toți descendenții unui membru al familiei?

7. Cum procedăm pentru a determina cel mai vârstnic ascendent al familiei?

5.4. Prelucrarea imaginilor

Aplicația 3. Virusuri într-o matrice

După ce au descoperit cele mai puternice virusuri, biologii și-au propus să le izoleze și să le studieze comportamentul în alte colonii de bacterii. După o vreme, au constatat că unele virusuri au rămas izolate, altele nu. Privită la microscop, colonia de bacterii pare formată dintr-o mulțime de puncte pe care biologii le-au reprezentat ca în figura de mai jos.

În această colonie există un singur virus izolat (cel încercuit). Biologii doresc să prelucreze aceste „imagini" cu calculatorul. Pentru început vor să localizeze și să numere virusurile izolate.

Cum rezolvăm problema?

1. Introducerea imaginii

De data aceasta, datele de intrare au aspectul unei „imagini alb-negru": punctele de interes sunt virusurile colorate în negru; restul suprafeței este colorată în alb. Pentru a transmite calculatorului „imaginea", ar trebui să-i spunem culoarea fiecărui punct. Sau, mai simplu, să-i spunem de la început că imaginea este „albă" și apoi să-i dăm adresele punctelor colorate în „negru". Imaginea descompusă în puncte seamănă atât de bine cu o matrice, încât ea poate fi memorată codificând albul cu zero și negrul cu unu.

	x				x
x			x	x	x
			x	x	
	(x)		x		

0	1	0	0	0	1
1	0	0	1	1	1
0	0	0	1	1	0
0	1	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Localizarea virusurilor izolate

nembru al familiei.

tru exemplul analizat, 1 are copii

niliei	Căsătorit cu	Copii
	3	1, 5, 7
	8	1, 10
	4	9

mbru al familiei?

t al familiei?

și-au propus să le izoleze și să
vreme, au constatat că unele
lonia de bacterii pare formată
în figura de mai jos.
). Biologii doresc să prelucreze
și să numere virusurile izolate.

alb-negru": punctele de intere
alb. Pentru a transmite calcu
t. Sau, mai simplu, să-i spunem
punctelor colorate în „negru"
ce, încât ea poate fi memorat

0	0	0	1
0	1	1	1
0	1	1	0
0	1	0	0
0	0	0	0
0	0	0	0

2. Reprezentarea algoritmului

Secvența de operații pentru citirea imaginii:

```
început citire
   alocă M[10,10]
   citește NL, NC    //număr linii -NL
                     //număr coloane -NC
   pentru i = 1, NL execută
      pentru j = 1, NC execută
         M[i, j] ← 0
      sfârșit pentru
   sfârșit pentru
   citește p //numărul de puncte negre
   pentru k = 1,p execută
   bloc
      citește l,c
      M[l,c] ← 1
   sfârșit bloc
   sfârșit pentru
sfârșit citire
```

3. Prelucrarea imaginii. Bordajul

Urmează prelucrarea „imaginii": localizarea și numărarea virusurilor (punctele negre izolate). Pentru rezolvarea acestei cerințe, se parcurge matricea și, pentru fiecare valoare de 1, se verifică vecinii acesteia. Un punct „negru" este izolat dacă toți vecinii săi sunt „albi".

Câți vecini are un punct? Majoritatea punctelor au câte opt vecini. Punctele de la „bariera" coloniei au mai puțini vecini.

Pentru a simplifica procedeul de numărare a vecinilor, vom înconjura colonia cu o zonă fără virusuri. Imaginea se lărgește. La fel și matricea: în urma acestui „bordaj", ea se va mări cu două linii și două coloane.

0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	1	1	1	0
0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Bordajul se pregătește înainte de citirea imaginii prin alocarea unui spațiu de memorie suficient, ținând seama că la numărul de linii și coloane necesar memorării imaginii se mai adaugă câte două pentru bordaj.

În matricea bordată, fiecare punct al imaginii are câte opt vecini. În figura 16 sunt prezentate adresele vecinilor punctului aflat la adresa **[l, c]**.

l-1, c-1	l-1, c	l-1, c+1
l, c-1	l, c	l, c+1
l+1, c-1	l+1, c	l+1, c+1

Figura 16. Vecinii unui punct din matrice

Dacă punctul aflat la adresa [**l, c**] are valoarea 1 și fiecare dintre cei opt vecini ai săi are valoarea zero, atunci punctul este izolat. Condiția o vom nota cu **cond**.

4. Reprezentarea algoritmului

• Secvența pentru localizarea și numărarea punctelor izolate:

```
început puncte_izolate
    nrp ← 0
    pentru l = 2,NL + 1 execută
        pentru c = 2,NC+1 execută
            dacă M[l,c] = 1 atunci
                dacă cond
                atunci
                    bloc    scrie l - 1, c - 1
                            nrp ← nrp + 1
                    sfârșit bloc
                sfârșit dacă
            sfârșit dacă
        sfârșit pentru
    sfârșit pentru
    scrie nrp
sfârșit puncte_izolate
```

TEMĂ

Folosind cele două secvențe prezentate, realizați algoritmul complet și programul corespunzător

De reținut!

1. Cu ajutorul matricelor pot fi rezolvate probleme care necesită descompunerea unei imagini în puncte sau probleme de orientare și deplasare în plan.

2. Pentru ușurința prelucrării punctelor se folosește tehnica bordajului.

3. Multe jocuri precum „X și zero", „Vaporașe", „Perspico", mutarea pieselor pe tabla de șah, „Război" pot fi simulate cu ajutorul matricelor.

PROBLEME PROPUSE

1. JOC

Se dau *n* jetoane numerotate de la 1 la *n*. Din fiecare tip de jeton, există *n* bucăți. Jetoanele trebuie așezate pe o grilă formată din *n*n* careuri, ca în exemplul din figura 17 a.

Figura 17. *Joc cu jetoane*

a)

1	2	3	4	5	6
2	3	4	5	6	1
3	4	5	6	1	2
4	5	6	1	2	3
5	6	1	2	3	4
6	1	2	3	4	5

b)

1	2	3	4	5
2	3	4	5	1
3	4	4	1	2
4	5	1	2	3
5	1	2	3	4

Cerințe:

1. Stabiliți regula după care sunt așezate jetoanele pe grila de joc.

2. Realizați un program care să așeze jetoanele pe grila de joc după regula stabilită la cerința precedentă.

3. Realizați un program care să verifice dacă jetoanele de pe grila de joc respectă regula de amplasare stabilită la cerința 1.

Spre exemplu, pentru grila de joc din figura 17 b, nu sunt îndeplinite următoarele proprietăți:
- pe diagonala secundară se află un jeton diferi de jetonul *n*;
- pe linia 3 nu se află jetoane distincte;
- pe coloana 3 nu se află jetoane distincte.

Indicație: se vor analiza jetoanele dispuse pe aceeași linie/coloană sau pe diagonale/paralele la diagonale.

2. TABELE MATEMATICE

a) TABLA ÎNMULȚIRII

Realizați un program pentru afișarea sub formă de tabel a tablei înmulțirii cu 1, 2, 3, până la 10. Spre exemplu, linia 3 va conține tabla înmulțirii cu 3:

3	6	9	12	15	18	21	24	27	30

b) TABLA ADUNĂRII

Realizați un program pentru afișarea sub forma de tabel a tablei adunării cu 1, 2, 3, până la 10. Spre exemplu, linia 7 va conține tabla adunării cu 7:

8	9	10	11	12	13	14	15	16	17

c) TABELA PITAGORA_n

Realizați un program pentru afișarea tabelei *Pitagora_n* sub formă de tabel cu *n* linii și trei coloane: *a, b, c*. Pentru fiecare linie, valorile din tabelă trebuie să respecte condiția: *a*a = b*b + c*c*. **Exemplu:** tabela *Pitagora_2*

5	3	4
15	12	9

d) MATRICEA *n_PALINDROM*

Realizați un program care să verifice dacă o matrice este *n_Palindrom*. O matrice *n_Palindrom* are *n* linii; pe fiecare linie, *i*, se află câte *n* palindromuri distincte formate din *i* cifre. Un număr simetric se numește *palindrom*. Exemplu de matrice *5_Palindrom:*

1	9	7	6	3
22	33	66	4455	55
121	323	464	585	979
3223	8668	7337	8118	9009
56365	43234	11111	91219	37473

3. MATRICE RARĂ

O matrice cu *n* linii și *m* coloane se numește *matrice rară* dacă valorile 0 sunt majoritare. **Exemplu** pentru o matrice cu 3 linii și 5 coloane:

0	0	20	0	0
0	0	0	12	0
0	88	0	0	0

Într-o astfel de matrice, interesează valorile semnificative, de aceea este suficient să memorăm valorile diferite de zero și pozițiile lor în matrice. Poziția reprezintă în acest caz numărul de ordine de 1 la *n*m*.

Pentru exemplul prezentat, se vor memora perechile: (20, 3) , (12, 9) și (88,12).

Cerințe:

a) propuneți structurile de date necesare memorării valorilor semnificative dintr-o matrice rară;

b) se cunoaște numărul valorilor semnificative dintr-o matrice rară (fie *p* acest număr) și perechile *valoare, adresă* pentru fiecare număr semnificativ.

Scrieți un program care să genereze și să afișeze pe ecran matricea rară.

Exemplu: Date de intrare
n =4 m=4 p=2
23 4 9 14

Date de ieșire — matricea rară

0	0	0	23
0	0	0	0
0	0	0	0
0	9	0	0

gora_n sub formă de tabel cu *n* linii ș
lă trebuie să respecte condiția:

trice este *n_Palindrom*. O matrice
palindromuri distincte formate din
de matrice *5_Palindrom:*

	3
	55
	979
	9009
9	37473

ară dacă valorile 0 sunt majoritare.

	0
	0
	0

e, de aceea este suficient să memo-
reprezintă în acest caz numărul de

3), (12, 9) și (88,12).

rilor semnificative dintr-o matrice

natrice rară (fie *p* acest număr) și

matricea rară.

natricea rară

0	0	0	23
0	0	0	0
0	0	0	0
0	9	0	0

6. ORGANIZAREA DATELOR ÎN STRUCTURI NEOMOGENE

6.1. Studiu de caz *Catalogul clasei*

Într-o clasă sunt 30 de elevi. Profesorul diriginte dorește să păstreze, într-un catalog electronic, următoarele informații (figura 18):

- ***numele elevului;***
- ***media generală pe semestrul I;***
- ***media generală pe semestrul II;***
- ***media generală anuală;***
- ***numărul de absențe.***

Profesorul diriginte dorește să calculeze media generală a clasei la sfârșit de semestru sau an școlar și să afișeze:

- ***media generală a clasei;***
- ***lista elevilor după media generală;***
- ***lista elevilor după numărul de absențe.***

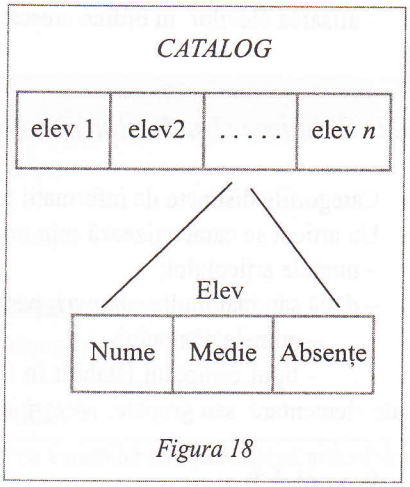


Figura 18

Analiza problemei

În catalogul electronic se vor păstra date despre elevi; se poate spune că în catalogul electronic se vor păstra date cu aceeași semnificație. Numărul înregistrărilor din catalog va fi egal cu numărul de elevi din clasă. Se va ține seama de numărul maxim de elevi ce pot fi înscriși într-o clasă (30 de elevi).

Catalogul poate fi organizat ca un tablou unidimensional – vector de elevi.

Pentru fiecare elev se rețin mai multe categorii de date, cu semnificații diferite – *date neomogene*. Fiecare informație are un tip specific, spre exemplu:

- medie de tip *real*;
- număr de absențe de tip *întreg* cu valori pozitive.

Soluția de organizare a datelor

Ansamblul datelor despre *elev* formează o structură de date cu tipuri diferite (structură neomogenă) numită *articol* sau *înregistrare*.

Catalogul electronic al clasei va fi un *vector de articole*.

Prelucrarea datelor

Pentru rezolvarea cerințelor formulate de profesorul diriginte, sunt necesare următoarele prelucrări:

- înregistrarea datelor pentru fiecare elev;
- afișarea datelor despre elevi;
- determinarea și afișarea mediei generale a clasei la sfârșit de semestru;

– afișarea elevilor în ordine descrescătoare, după media generală semestrială;
– determinarea mediei generale, anuale, pentru fiecare elev;
– afișarea elevilor în ordine descrescătoare, după media generală anuală;
– determinarea și afișarea mediei generale a clasei la sfârșit de an;
– afișarea elevilor în ordine crescătoare, după numărul de absențe.

6.2. Definirea structurilor neomogene de date-articole

Categoriile distincte de informații reținute într-un articol se numesc *câmpuri*.
Un articol se caracterizează prin următoarele elemente ce formează *macheta articolului*:
– numele articolului;
– două sau mai multe *câmpuri*; pentru fiecare *câmp* se precizează:
– numele câmpului;
– tipul câmpului (stabilit în funcție de semnificația reală a datelor). Câmpurile pot
date elementare sau grupate: *vectori*, *articole*.

Exemplul 1:
Pentru exemplul analizat – catalogul clasei –, construim o machetă simplificată (cu m
puține câmpuri) corespunzătoare articolului *elev* (fig. 19a).

Elev	
Numele câmpului	Tipul câmpului
MEDIE	Real
ABSENȚE	Natural

Figura 19, a)

Exemplul 2:
Completăm macheta articolului *elev* cu câmpul note, un tablou unidimensional (*vector*) c
4 elemente de tip natural (fig. 19b).

Elev	
Numele câmpului	Tipul câmpului
NOTE	Vector (4)
MEDIE	Real
ABSENȚE	Natural

Figura 19, b)

Fiecare limbaj de programare dispune de cuvinte cheie pentru declararea structurilor de date de tip articol (Tabelul 12).

Tabelul 12. **Definirea articolelor**

PASCAL	C/C++
{cuvântul cheie pentru tipul articol este **record**}	// *cuvântul cheie pentru tipul articol* // *este* ***struct***
Type **articol = record** camp1:*tip1*; camp2:*tip2*; ..... end;	*Typedef* struct { *tip1* camp1; *tip2* camp2; ..... } **articol**;
{definirea variabilei cu tipul asociat articolului} *var* variabila : **articol**;	//definirea variabilei cu tipul asociat articolului **articol** variabila;
Exemplu pentru articolul elev	

{ se definește articolul elev }
Type elev=record
.....
medie:*real*;
absente:*byte*;
.....
end;
{se definește variabila cu tipul elev}
var e: elev;
	// se definește articolul elev
	Typedef struct
	{
	.....
	float medie;
	int:absente;
	.....
	} elev;
	// se definește variabila cu tipul elev
	elev e;

TEME

1. Realizați macheta articolului *data calendaristică* în care să fie reținută data în forma *zi, lună, an.*
2. Realizați macheta articolului *elev* care să conțină și câmpul *data nașterii* sub forma grupului de date *data calendaristică.*
3. Realizați macheta articolului *elev* care să conțină câmpul *medii* sub forma unui vector.

6.3. Prelucrarea datelor organizate în structuri neomogene

Prelucrarea datelor înregistrate într-o structură de tip articol se face la nivel de câmp. Pentru accesul la informațiile unui câmp, referirea acestuia se face prin *adresare punctuală* după sintaxa: *nume-variabilă-de-tip-articol.nume-câmp.*

Exemplu:

– referirea câmpului medie din articolul *elev* **e.medie**
– referirea câmpului absențe din articolul *elev* **e.absente**
– referirea unui element, i, al câmpului medii din articolul *elev* **e.medii[i]**

Prelucrarea acestor date începe cu introducerea și memorarea lor în zona de memorie rezervată la definirea variabilei de tip articol; pentru verificare, datele memorate vor fi afișate. Fiecare câmp poate fi prelucrat prin operații specifice tipului corespunzător (Tabelul 13).

Tabelul 13. **Prelucrarea articolelor**

PASCAL	C / C++
Exemplu pentru articolul elev	
1. Introducerea și afișarea datelor	

Program exemplu;
{ se defineste articolul elev }
Type elev=record
. . . . . .
medie1, medie2, medie_an:*real*;
absente:*byte*;
. . . . . .
end;
 	
{se defineste variabila cu tipul elev}
 	
var e: *elev*;
 	
begin
 	
{se introduc datele unui elev}
write ('medie1:'); readln (e.medie1);
write ('medie2:'); readln (e.medie2);
write ('absente:'); readln (e.absente);
. . . . . .
{se afiseaza datele elevului}
writeln ('medie1:', e.medie1);
writeln ('medie:2', e.medie2);
writeln ('absente:', e.absente);
 	
end.
	#include <iostream.h>
	void main ()
	{// se defineste articolul elev
	Typedef struct
	{
	. . . . . .
	float medie1, medie2, medie_an;
	unsigned:absente;
	. . . . . .
	} elev;
	
	// se defineste variabila cu tipul elev
	
	elev e;
	
	// se introduc datele unui elev
	cout<< " medie1:"; cin>> e.medie1;
	cout<< " medie2:"; cin>> e.medie2;
	cout<< " absente:"; cin>> e.absente;
	. . . . . .
	// se afiseaza datele elevului
	
	cout<< " medie1:"<<e.medie1<<endl;
	cout<< " medie2:"<<e.medie2<<endl;
	cout<< " absente:"<< e.absente<<endl;
	}

col se face la nivel de câmp. Pentru
prin *adresare punctuală* după sin-

l *elev* e.medii[i]
area lor în zona de memorie rezer-
ele memorate vor fi afișate. Fiecare
nzător (Tabelul 13).

C / C++
lev
elor
stream.h>
te articolul elev
t
. . . . .
iel, medie2, medie_an;
absente;
. . . . .
variabila cu tipul elev
datele unui elev
el:"; cin>> e.medie1;
e2:"; cin>> e.medie2;
nte:"; cin>> e.absente;
. . . . .
atele elevului
e1:"<<e.medie1<<endl;
e2:"<<e.medie2<<endl;
te:"<< e.absente<<endl;

2. Operații asupra datelor	

{ se calculează media anuală}
e.medie_an := (e.medie1 +e.medie2)/2;
 	
{se afiseaza media anuala}
writeln('medie anuala: ', e.medie_an);
	// se calculează media anuală
	e.medie_an= (e.medie1 +e.medie2)/2;
	
	// se afiseaza media anuala
	cout<<:"medie anuala:"<< e.medie_an;

3. Introducerea și afișarea datelor în câmpuri grupate	

Type elev=record
. . . . . .
medii : array [1..3] of *real*;
. . . . . .
end;
 	
{se defineste variabila cu tipul elev}
var e: *elev*; i:byte;
begin
{se introduc mediile unui elev}
for i:=1 to 3 do
begin
write ('media:',i);
readln (e.medii[i]);
end;
end
	Typedef struct
	{
	. . . . . .
	float medii [3];
	. . . . . .
	} elev;
	// se defineste variabila cu tipul elev
	elev e;
	int i;
	{
	// se introduc mediile unui elev
	for (i=0; i<3;i++)
	{cout<< " media:"<<i;
	cin>> e.medii[i];
	}
	}

TEME

1. Descrieți sub formă de articol următoarele entități:
   a) carte; b) mașină; c) calculator; d) persoană.
2. Definiți, în limbajul de programare studiat, tipurile de date și variabilele corespunzătoare articolelor descrise la cerința precedentă.
3. Realizați, în limbajul de programare studiat, un program care să determine trimestrul calendaristic al zilei curente. Se va folosi un articol cu câmpurile: *zi, lună, an.*
   Exemplu: pentru data curentă: ziua 29 luna 08 anul 2007, se va afișa: trimestrul 3.
4. Realizați, în limbajul de programare studiat, un program care să determine dacă două puncte *A* și *B* îndeplinesc una dintre următoarele condiții:
   a) punctele A și B se află pe o dreaptă paralelă cu axa OX;
   b) punctele A și B se află pe o dreaptă paralelă cu axa OY;
   c) punctele A și B se află pe o dreaptă egal depărtată de axele OX si OY.
   Pentru fiecare punct se cunosc coordonatele *x* și *y*. Fiecare punct va fi descris printr-un articol.
   Exemplu: Pentru punctul A de coordonate x=3, y=10 și punctul B de coordonate x=35, y=10 se va afișa mesajul: *punctele se află pe o dreaptă paralelă cu axa OX.*

6.4. Gruparea datelor organizate în structuri neomogene

Datele cu semnificații diferite care necesită gruparea în structuri neomogene reprezintă, în cele mai frecvente cazuri, caracteristici sau atribute ale unei situații din realitate (entitate): persoană, produs, carte, mașină etc. Memorarea datelor prin care poate fi descrisă o entitate se face într-o singură zonă de memorie (variabilă); dacă în problemă sunt prelucrate date despre mai multe *instanțe* ale unei *entități* (mai multe persoane, mai multe produse), se vor păstra întotdeauna datele ultimei instanțe introduse (ultima persoană, ultimul produs).

Exemplu:

*La un magazin, se vând într-o zi **n** produse. Se cunosc cantitatea și prețul fiecărui produs; se dorește calcularea și afișarea valorii totale a produselor vândute într-o zi.*

Rezolvarea problemei necesită introducerea, pe rând, a datelor despre fiecare produs. Pentru fiecare produs introdus, se calculează valoarea:

*valoare = cantitate * pret.*

Valoarea calculată se însumează la *total:*

total= total + valoare.

La sfârșit, se afișează *totalul.* (Tabelul 14)

Tabelul 14. **Exemplu *Magazin***

PASCAL	C/C++

program produse;
 	
{ se defineste articolul produs }
Type produs=record
cantitate:*byte*;
pret:*real*;
end;
{se defineste variabila cu tipul produs}
var p: *produs*;
{se definesc variabilele de lucru}
var valoare, total: real;
n, i :integer;
 	
begin
 	
{se introduce numarul de produse}
Write('numarul de produse:');
readln (n);
total:=0;
for i:= 1 to n do
begin
	#include <iostream.h>
	void main ()
	{
	// se defineste articolul produs
	Typedef struct
	{*unsigned* cantitate
	float pret;
	} produs;
	// se defineste variabila cu tipul produs
	produs p;
	//se definesc variabilele de lucru
	float valoare, total=0;
	
	int n, i ;
	
	//se introduce numarul de produse
	cout<< " numarul de produse:";
	cin>> n;
	for (i=1; 1<n; i++)
	{
	//se introduc datele unui produs

structuri neomogene reprezintă, în
situații din realitate (entitate): per-
e poate fi descrisă o entitate se face
nă sunt prelucrate date despre ma
e produse), se vor păstra întotdeau-
produs).

ntitatea și prețul fiecărui produs; se
ndute într-o zi.
telor despre fiecare produs. Pentru
t.

C/C++

stream.h>

e articolul produs
t
cantitate

e variabila cu tipul produs

variabilele de lucru
, total=0;

numarul de produse
arul de produse:";

i++)

datele unui produs

```
{se introduc  datele unui produs}
write('cantitate:');
readln (p.cantitate);
write ('pret:'); readln (p.pret);

{se calculeaza valoarea produsului}
valoare:= p.cantitate * p.pret;
{se insumează valoarea la total}
total:= total + valoare;
end;

{se afiseaza valoarea totala}
Writeln ('valoarea totala:', total);
end
```

```
cout<< " cantitate:";
cin>>p.cantitate;
cout<< " pret:"
cin>>p.pret;

//se calculeaza valoarea produsului
valoare = p.cantitate * p.pret;
//se insumează valoarea la total}
total= total + valoare;
}

//se afiseaza valoarea totala
cout<<"valoarea totala:"<<total;
}
```

În cele mai multe situații reale, se păstrează datele fiecărei instanțe (fiecare persoană, fiecare produs) pentru prelucrări la nivel de grup: sortări, selecții. În *Catalogul clasei*, se păstrează datele despre toți elevii.

Gruparea instanțelor unei entități (toate persoanele, toate produsele, toți elevii) se face prin memorarea valorilor corespunzătoare în tablouri unidimensionale care, în acest caz, devin *vectori de articole*. Un element al vectorului reprezintă o instanță (o persoană, un produs, un elev) și este de tip articol.

Câmpul unui articol dintr-un vector de articole se adresează punctual, cu precizarea că, în acest caz, numele articolului este înlocuit cu adresa acestuia în vector (Tabelul 15).

Tabelul 15. **Vectori de articole**

PASCAL	C/C++

{definirea tipului de date articol}
Type articol=record
camp1:*tip1*;
camp2:*tip2*;
. . . . . .
end;
{definirea vectorului cu n elemente de tipul asociat articolului}
var vector:array [1..10] of articol;
var i : byte;
{adresarea unui camp dintr-un articol
varecare,i}
vector [i].camp
	// definirea tipului de date articol
	Typedef struct
	{ *tip1* camp1;
	tip2 camp2;
	. . . . . .
	} articol;
	// definirea vectorului cu n elemente
	//de tipul asociat articolului
	articol vector [10];
	int i;
	//adresarea unui camp dintr-un articol
	//oarecare,i}
	vector [i].camp

Exemplul 1 *prelucrarea a **n** produse*

PASCAL

```
program produse;
{ se defineste articolul  produs }
Type produs=record
   cantitate:byte;
   pret:real;
  end;

{se defineste vectorul p cu  100 de elemente
de  tip produs}
var p:array [1..100] of  produs;

{se definesc variabilele de lucru}
var valoare, total: real;
   n, i :integer;
begin

{se introduce numarul de produse}
write('numarul de produse:');
readln (n);
total:=0;
for i:= 1 to n do
begin

{se introduc  si se memoreaza  datele fiecarui
produs}
write('cantitate:');
readln (p[i].cantitate);
write ('pret:'); readln (p[i].pret);

{se calculeaza valoarea produsului}
valoare:= p[i].cantitate * p[i].pret;
{se însumează valoarea la total}
total:= total + valoare;
end;
{se afiseaza valoarea totala}
writeln ('valoarea totala:', total);
end.
```

C/C++

```
#include <iostream.h>
void main ()
{
// se defineste articolul  produs
Typedef struct
   {unsigned cantitate
    float pret;
    } produs;

// se defineste vectorul p cu 100
//elemente de tip produs
produs  p[100];

//se definesc variabilele de lucru
float  valoare, total=0;
int    n, i ;

//se introduce numarul de produse
cout<< " numarul de produse:";
cin>> n;
for (i=0; i<n; i++)
{

//se introduc si se memoreaza  datele //
fiecarui produs
cout<< " cantitate:";
cin>>p[i].cantitate;
cout<< " pret:"
cin>>p[i].pret;

//se calculeaza valoarea produsului
valoare = p[i].cantitate * p[i].pret;
{se însumează valoarea la total}
total= total + valoare;
}
//se afiseaza valoarea totala
cout<<"valoarea totala:"<<total;
}
```

Exemplul 2 sortarea a ***n*** produse - metoda *Bubble sort*

```
var aux: produs; s:byte;
begin
{sortare crescatoare dupa campul pret}
repeat
 s:=0;
for i:=1 to n-1 do
 begin
   if p[i].pret> p[i+1].pret
     then
{interschimbarea se face la nivel de articol }
       begin
         aux:=p[i];
          p[i]:= p[i+1];
          p[i+1]:= aux;
          s:=1;
       end;
until s=0;
writeln (' lista preturilor');
for i:=1 to n do
writeln (p[i].pret);
end
```

```
{produs aux; unsigned s;
//sortare crescatoare dupa campul pret
do
{
  s=0;
  for (i=0; i< n-1; i++)
  if p[i].pret> p[i+1].pret

//interschimbarea se face la nivel de articol
        {
          aux=p[i];
           p[i]= p[i+1];
           p[i+1]= aux;
           s=1;
        }
}
while (s);
cout<<"lista preturilor"<<endl;
for (i=0; i< n; i++)
cout<<p[i].pret<<endl;
}
```

TEME

1. Formulați exemple care să necesite organizarea datelor în vectori de articole.
2. Prezentați o situație reală care să necesite ordonarea datelor organizate în vectori de articole.
3. Alcătuiți o listă cu prelucrări specifice datelor grupate în tablouri unidimensionale; pentru fiecare prelucrare, construiți câte un exemplu care să necesite gruparea articolelor în vectori de articole.

De reținut! ARTICOL

- **structură de date necesară pentru înregistrarea informațiilor despre un aspect al realității (obiect, persoană, activitate) cu mai multe caracteristici;**
- **fiecare caracteristică formează un câmp al articolului;**
- **fiecare câmp poate avea un tip propriu; acest aspect determină proprietatea de structură neomogenă;**
- **memorarea și prelucrarea datelor organizate în structuri neomegene se face la nivel de câmp;**
- **referirea unui câmp se face prin adresare punctuală:**
  - **nume_articol.nume_câmp;**
- **articolele pot fi grupate în structuri omogene: vectori de articole;**
- **referirea unui câmp de articol dintr-un vector se face prin adresare punctuală, înlocuindu-se numele articolului cu adresa acestuia în vector.**

PROBLEME PROPUSE

1. Colecție

Se dorește înregistrarea următoarelor date despre obiectele dintr-o colecție: denumire, an achiziției, valoare.

Cerințe:

a) realizați un program pentru introducerea și afișarea datelor despre un obiect din colecție

b) realizați un program pentru introducerea și afișarea datelor despre mai multe obiecte di colecție (max. 100);

c) realizați un program care să determine cel mai vechi obiect din colecție;

d) realizați un program care să afișeze obiectele din colecție în ordinea descrescătoare a vechimii acestora.

2. Concurs

Se dorește înregistrarea următoarelor date despre candidații înscriși la un concurs: numele data nașterii (zi, lună, an), are carnet de conducere (da/nu).

Cerințe:

a) realizați un program pentru introducerea și afișarea datelor despre un singur candidat;

b) realizați un program pentru introducerea și afișarea datelor despre mai mulți candidați (max. 100);

c) realizați un program care să determine candidații cu aceeași vârstă, *v*, introdusă de la tastatură;

d) realizați un program care să afișeze, în ordinea vârstei, candidații cu carnet de conducere.

3. Catalogul clasei

Realizați un program cu meniu care să rezolve următoarele cerințe ale profesorului diriginte:

- înregistrarea datelor pentru fiecare elev;
- afișarea datelor despre elevi;
- determinarea și afișarea mediei generale a clasei la sfârșit de semestru;
- afișarea elevilor în ordine descrescătoare, după media generală semestrială;
- determinarea mediei generale, anuale, pentru fiecare elev;
- afișarea elevilor în ordine descrescătoare, după media generală anuală;
- determinarea și afișarea mediei generale a clasei la sfârșit de an;
- afișarea elevilor în ordine crescătoare, după numărul de absențe.

7. ORGANIZAREA DATELOR ÎN STRUCTURI DINAMICE

7.1. Modele de structuri dinamice

În foarte multe situații reale, există relații sau reguli care trebuie modelate astfel încât soluția de organizare a datelor să respecte atât semnificația, cât și restricțiile de comportament specifice.

obiectele dintr-o colecție: denumire, anu

area datelor despre un obiect din colecție;
șarea datelor despre mai multe obiecte di

vechi obiect din colecție;
din colecție în ordinea descrescătoare

candidații înscriși la un concurs: numele
a/nu).

area datelor despre un singur candidat;
işarea datelor despre mai mulți candidați

ții cu aceeași vârstă, *v*, introdusă de la tas-

vârstei, candidații cu carnet de conducere

ătoarele cerințe ale profesorului diriginte

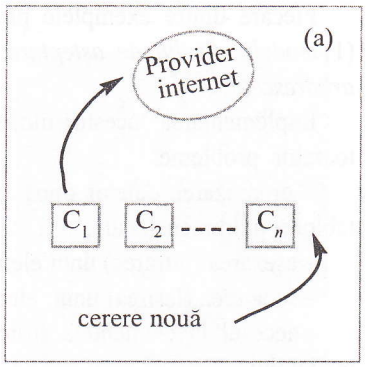
la sfârșit de semestru;
media generală semestrială;
care elev;
media generală anuală;
la sfârșit de an;
nărul de absențe.

TURI DINAMICE

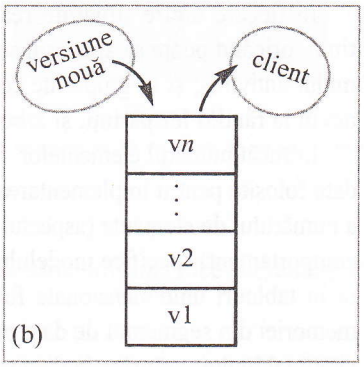
i care trebuie modelate astfel încât soluția
cât și restricțiile de comportament spe-

Exemple:

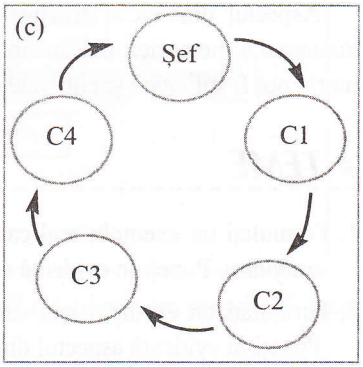
Exemplul 1. Persoanele care solicită un serviciu – conectarea la Internet printr-un provider autorizat – se înscriu pe o listă de așteptare. O cerere nouă este așezată, întotdeauna, ultima în listă. Serviciul de conectare este acordat, întotdeauna, primului solicitant din listă; după acordarea serviciului, cererea este eliminată din listă (fig. 20 a).



Exemplul 2. O firmă de software a realizat un program antivirus; programul este îmbunătățit continuu, prin tratarea de noi viruși; oferta de piață a firmei este organizată astfel încât clienții să aibă acces la program, începând întotdeauna cu ultima versiune a acestuia (fig. 20 b).



Exemplul 3. Pentru a comunica rapid și sigur, „șeful" unui grup de copii a întocmit o listă astfel încât un mesaj să poată fi transmis întregului grup, din copil în copil; „șeful" transmite mesajul primului copil din listă; ultimul copil comunică „șefului" că mesajul a ajuns la el (fig. 20 c).



4. Fiecare persoană are exact doi părinți; fiecare părinte este o persoană care la rândul ei are exact doi părinți. Pentru a păstra atât datele despre persoane, cât și relațiile directe de rudenie copil-părinți, se construiește arborele de familie – arborele genealogic (fig. 20 d).

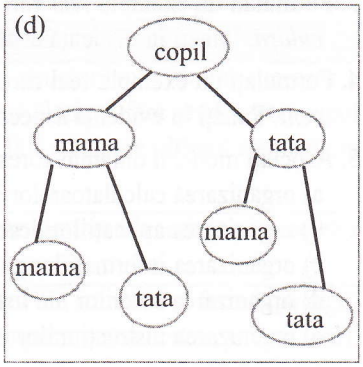


Figura 20

Fiecare dintre exemplele propuse necesită organizarea datelor după un model propr (1) *modelul firului de așteptare*; (2) *modelul stivei*; (3) *modelul listei circulare*; (4) *mode arborescent*.

Implementarea acestor modele într-un limbaj de programare necesită soluționarea urm toarelor probleme:

– organizarea datelor după semnificația acestora, cel mai frecvent în structuri omoge tablouri unidimensionale;

– așezarea (intrarea) unui element din structură după regula specifică modelului;

– scoaterea (ieșirea) unui element din structură după regula specifică modelului;

– accesul la elementele structurii (numărarea/listarea elementelor) după regula specifi modelului.

În fiecare dintre situațiile reale din exemplele prezentate, numărul elementelor variază timp: oricând poate să apară un solicitant pentru serviciul Internet sau o versiune nouă a progr mului antivirus; și în grupul de copii poate intra sau poate pleca un copil; în orice familie, cop devin la rândul lor părinți, și arborele genealogic crește.

Întrucât numărul elementelor nu este constant și nici nu poate fi precizat în timp, structurile date folosite pentru implementarea acestor modele se numesc *structuri dinamice*. Variația în tim a numărului de elemente (aspectul dinamic al structurii) respectă relațiile și disciplina (regulile comportament) specifice modelului. Implementarea structurilor dinamice prin memorarea acest ra în tablouri unidimensionale folosește *alocarea statică* a memoriei (mecanism de alocarea memoriei din segmentul de date prin care zona de memorie maxim alocată – corespunzător cap cității tabloului – rămâne la dispoziția programului pe toată durata de execuție a acestuia).

Aspectul dinamic al structurilor de date este pus și mai bine în evidență în modul de *aloca dinamică* a memoriei: mecanism de alocarea a memoriei de tip *Heap* prin care zonele de me morie pot fi solicitate și eliberate chiar în timpul execuției programului.

TEME

1. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip *fir a așteptare*. Puneți în evidență aspectul dinamic al structurii.
2. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip stiv Puneți în evidență aspectul dinamic al structurii.
3. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip *listă ci cular*ă. Puneți în evidență aspectul dinamic al structurii.
4. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip *arbore cent*. Puneți în evidență aspectul dinamic al structurii.
5. Asociați modelul dinamic corespunzător fiecăreia dintre următoarele situații:
   a) organizarea calculatoarelor într-o rețea locală de tip inel;
   b) organizarea aplicațiilor deschise de un utilizator în sistemul de operare Windows;
   c) organizarea informațiilor pe discul sistem;
   d) organizarea cererilor de listare la imprimantă;
   e) organizarea instrucțiunilor unui program.

ARBORELE GENEALOGIC
temă de compoziție

Realizați arborele genealogic personal printr-o prezentare cât mai atractivă care să pună în evidența personalitatea fiecărui membru al familiei.

Se poate lucra în oricare dintre aplicațiile cu efecte grafice cunoscute.

Sugestie de rezolvare:

– prezentarea acestei teme de către elevi, în laborator, conduce la o activitate foarte atractivă; se pot face „clasamente": cel mai „înalt" arbore; cel mai „vârstnic" arbore, cel mai „stufos" arbore.

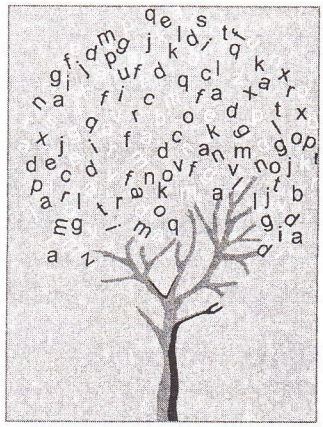
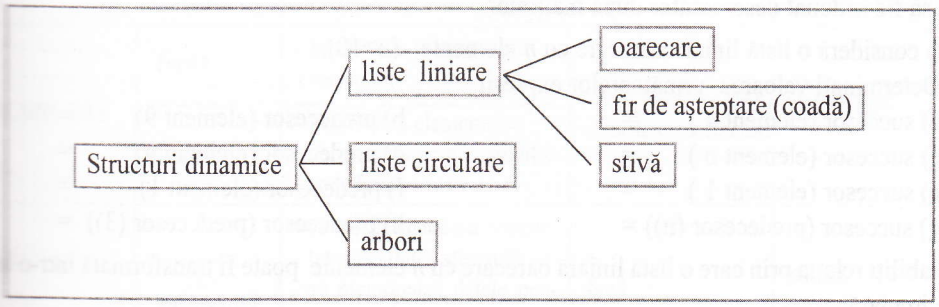


Figura 21

7.2. Clasificarea structurilor dinamice

Relațiile dintre elemente determină mai multe categorii de structuri dinamice prezentate în clasificarea din figura 22.

Figura 22. Clasificarea structurilor dinamice



Fiecare dintre categoriile de structuri dinamice din această clasificare are proprietăți specifice (Tabelul 16).

Tabelul 16. **Proprietățile structurilor dinamice**

STRUCTURA DINAMICĂ	PROPRIETĂȚI
Listă liniară	Relațiile dintre elementele structurii sunt de tip succesor – predecesor; există un singur element fără predecesor, capul listei, și un singur element fără succesor, ultimul element din listă.
Listă oarecare	Nu există nicio regulă pentru intrarea și ieșirea elementelor din structură.
Fir de așteptare	Intrarea și ieșirea elementelor din structură se face după regula FIFO (First Input First Output = *primul intrat, primul ieșit*).

Stivă	Intrarea și ieșirea elementelor din structură se face după regula LIFO (Last Input First Output = *ultimul intrat, primul ieșit*).
Listă circulară	Relațiile dintre elementele structurii sunt de tip succesor – predecesor; nu există niciun element fără predecesor sau fără succesor.
Arbori	Relațiile dintre elementele structurii sunt de tip ascendent – descendent; există un singur element fără ascendent – rădăcina arborelui – și unul sau mai multe elemente fără descendenți – elementele terminale sau frunze.

TEME

1. Identificați tipul listei liniare care poate fi asociat următoarelor situații:

a) Profesorul diriginte întocmește lista cu elevii care vor participa la o excursie cu număr limitat de locuri (mai puține decât efectivul clasei).

b) Întrucât numărul elevilor care doresc să meargă în excursie este mult mai mare decât numărul de locuri, profesorul diriginte reface lista pentru a-i elimina mai ușor pe cei care s-au înscris mai târziu.

c) La ora de Educație Fizică, elevii intră pe rând din vestiar în sala de sport; fiecare elev trebuie să se alinieze, ocupându-și locul astfel încât, în fiecare moment, șirul elevilor prezenți în sală să fie ordonat descrescător după înălțime.

2. Se consideră o listă liniară oarecare cu *n* elemente (n>10).

Determinați valoarea următoarelor expresii:

a) succesor (element 4) = b) predecesor (element 9) =

c) succesor (element n) = d) predecesor (element n) =

e) succesor (element 1) = f) predecesor (element 1) =

g) succesor (predecesor (n)) = h) predecesor (predecesor (3)) =

3. Stabiliți relația prin care o listă liniară oarecare cu *n* elemente poate fi transformată într-o listă circulară.

4. 4.1. Pentru fiecare membru din arborele genealogic personal, determinați:

a) numărul descendenților;

b) numărul ascendenților;

c) numărul elementelor terminale (fără descendenți).

4.2. Precizați care este semnificația elementelor terminale din arborele genealogic personal.

4.3. Care este semnificația elementului rădăcină din arborele genealogic personal?

4.4. Cum ar trebui construit arborele genealogic personal, astfel încât autorul să fie un element terminal?

telor din structură se face după regula
utput = *ultimul intrat, primul ieșit*).

ele structurii sunt de tip succesor –
iun element fără predecesor sau fără

le structurii sunt de tip ascendent –
singur element fără ascendent –
unul sau mai multe elemente fără
terminale sau frunze.

arelor situații:
participa la o excursie cu număr li-

excursie este mult mai mare decât
-i elimina mai ușor pe cei care s-au

iar în sala de sport; fiecare elev tre-
moment, șirul elevilor prezenți în

).

decesor (element 9) =
decesor (element n) =
ecesor (element 1) =
decesor (predecesor (3)) =
te poate fi transformată într-o listă

onal, determinați:

din arborele genealogic personal.
rele genealogic personal?
, astfel încât autorul să fie un ele-

7.3. Prelucrări specifice structurilor dinamice liniare

Pentru organizarea datelor în structuri dinamice liniare, sunt necesare următoarele prelucrări:

- **Crearea** structurii dinamice: această prelucrare corespunde memorării datelor pentru primul element din structură.
- **Parcurgerea** structurii dinamice: această prelucrare permite localizarea fiecărui element din structură, respectându-se regulile de ordine specifice modelului de structură dinamică.
- **Actualizarea** structurii dinamice: această prelucrare permite modificarea numărului de elemente din structură prin adăugare sau inserare de elemente noi sau prin eliminarea unor elemente; tot prin actualizare, se pot modifica informațiile specifice unui element.

În Tabelul 17 sunt descrise operațiile necesare implementării acestor prelucrări.

Tabelul 17. **Structuri dinamice – prelucrări și operații specifice**

PRELUCRARE		OPERAȚII SPECIFICE	STAREA STRUCTURII (numărul n de elemente)	
			ÎNAINTE DE PRELUCRARE	DUPĂ PRELUCRARE
CREARE		– se verifică dacă structura este vidă – se memorează datele pentru primul element din structură	dacă n=0 (structura este vidă)	*atunci* n=1 *altfel* *operație fără sens*
PARCURGERE	Totală	– localizarea tuturor elementelor	Nu se modifică nici numărul, nici valorile elementelor.	
	Parțială	– localizarea elementelor care îndeplinesc o condiție specificată		
ACTUALIZARE	Adăugare / Inserare	– se verifică dacă s-a completat capacitatea structurii – se memorează datele pentru un element nou care intră în structură respectând regula de intrare specifică modelului.	**dacă** n=capacitatea structurii	**atunci** *operație imposibilă* **altfel** n+1 elemente
	Ștergere	– se verifică dacă structura este vidă – se elimină din structură un element respectând regula de ieșire specifică modelului.	**dacă** n= 0 structura este vidă	atunci *operație imposibilă* **altfel** n-1 elemente
	Modificare	Se localizează (prin parcurgere parțială) elementul ale cărui valori trebuie modificate; se modifică valorile elementului localizat.	Nu se modifică numărul de elemente din structură; se modifică valorile pentru unul sau mai multe elemente.	

TEME

1. Precizați de ce este necesară cunoașterea capacității unei structuri dinamice în varianta implementării prin alocarea statică a memoriei.
2. Identificați situațiile în care pot fi date următoarele mesaje:
   a) *operație fără sens, listă existentă;*
   b) *operație imposibilă, listă vidă;*
   c) *operație imposibilă, element inexistent;*
   d) *nu este permisă operația de inserare.*
3. Alcătuiți câte o secvență de operații elementare, necesară fiecăreia dintre următoarele prelucrări:
   a) afișarea numărului de elemente dintr-o listă oarecare;
   b) intrarea unui element nou într-un fir de așteptare;
   c) afișarea numărului de ordine al elementelor care respectă o condiție specificată;
   d) verificarea existenței în listă a unui element care respectă o condiție specificată;
   e) ieșirea unui element dintr-o stivă.

7.4. Implementarea structurilor dinamice liniare

7.4.1. FIRUL DE AȘTEPTARE (COADA)

Disciplina structurii dinamice *fir de așteptare* sau *coadă* este de tip FIFO (First Input First Output = *primul intrat, primul ieșit*). Implementarea într-un limbaj de programare a acestui model de structură dinamică revine la controlul operațiilor de intrare/ieșire în/din structură, astfel încât să fie respectată disciplina FIFO.

În acest scop, structura va fi controlată prin:

(1) doi marcatori (indici) de poziție pe care îi vom numi *primul* și *ultimul* (*fig. 22*).

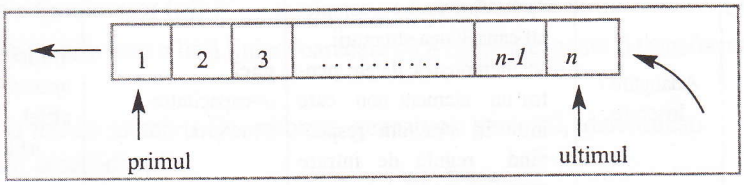


Figura 22

(2) *capacitatea structurii* (numărul maxim de elemente alocate pe care îl vom nota cu *n*).

Cazuri particulare

Dacă firul de așteptare nu conține niciun element, firul este gol (*fir vid*).

Dacă a fost ocupată toată capacitatea structurii, *firul este plin*.

Prelucrările specifice firului de așteptare

- **crearea** are sens doar dacă firul este gol (fir vid);
- **intrarea** unui element nou:
  - un element poate intra în structură, doar dacă nu a fost completată capacitatea structurii („mai sunt locuri libere");
  - întotdeauna, noul element se așază la sfârșitul structurii, el devine *ultimul*;
- **ieșirea** unui element (întrucât firele de așteptare/coada se formează pentru satisfacerea unor cereri, „servicii", spunem că ieșirea din fir are loc când *primul* element a fost *servit*):
  - un element poate ieși din structură, doar dacă firul nu este vid (există cel puțin un element);
  - întotdeauna, după ieșirea unui element din structură, toate elementele se deplasează „în față"; coada „avansează" și elementul ajuns pe poziția *primul* poate fi *servit (fig. 23);*

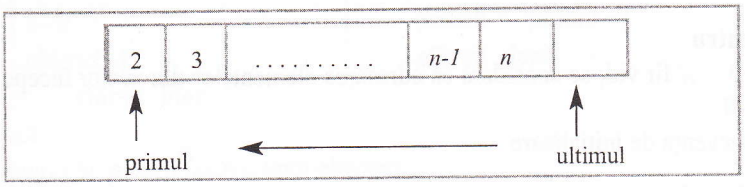


Figura 23

- **parcurgerea** se poate efectua doar dacă firul nu este vid:
  - parcurgerea se poate face pentru determinarea numărului de elemente „așezate la coadă", pentru afișarea elementelor sau pentru determinarea unor proprietăți;
  - parcurgerea se face întotdeauna de la *primul* la *ultimul* element.

Elementele firului de așteptare pot fi persoane, obiecte, procese care așteaptă satisfacerea unor cereri; exemple:

- persoanele înscrise pe o listă de așteptare pentru rezervări de bilete, acceptarea la un interviu etc.;
- persoanele care așteaptă la rând pentru efecturea unor plăți, cumpărarea de produse etc.;
- mașinile aflate într-un spațiu de parcare la o benzinărie;
- evenimentele planificate într-o locație (concerte, concursuri etc.).

În cele mai frecvente situații, se înregistrează mai multe informații despre elementele firului de așteptare. Fiecare element din fir este descris ca o structură eterogenă – articol; firul de așteptare va fi memorat într-un vector de articole.

Pentru început, vom considera că fiecare element este descris prin numărul său de ordine (numere cu valori întregi, pozitive).

În continuare, se prezintă secvența pseudocod a operațiilor necesare implementării unui fir de așteptare.

```
început fir _de_ așteptare_1
// se lucrează cu vectorul F pentru care se alocă 100 de elemente
// variabile de lucru:
// n numărul maxim de elemente din fir (capacitatea firului )
// primul ultimul marcatori de poziție
// i indicele de adresă pentru F
// nr numărul de elemente din fir

// secvența de inițializare
// secvența poate fi completată cu validarea lui n fața de numărul de elemente alocate (100)
scrie introduceti valoare pentru capacitatea firului de asteptare, n=
citește n
pentru i=1 la n execută
F[i] ← 0
sfârșit pentru
 primul←0    // fir vid; se consideră că adresarea elementelor din vector începe de la 1
 ultimul←0
// sfârșit secvența de inițializare

// secvența pentru creare
dacă primul = 0
atunci
 bloc
        primul←1
        scrie introduceti valoare pentru primul element F[primul]=
        citește F[primul]
        ultimul← primul
  sfârșit bloc
altfel
     scrie   operatie fara sens: firul nu este vid
sfârșit dacă
//sfârșit secvența creare

// secvența pentru intrarea unui element nou
ultimul ← ultimul+1
dacă ultimul < =n
     atunci
                bloc
          scrie introduceti valoare pentru noul element F[ultimul]=
citește F[ultimul]
          sfârșit bloc
        altfel
                  scrie   operatie imposibila: firul este plin
sfârșit dacă
//sfârșit secvența pentru intrarea unui element nou
```

```
// secvența pentru ieșirea (servirea) unui element
dacă primul =o
    atunci
            scrie   operatie imposibila:  firul este gol
      altfel
          bloc
    scrie   este servit elementul F[primul]

// coada avansează – secvența A
pentru i= primul  la ultimul-1 execută
      F [i] ← F[i+1]
 sfârșit  pentru
F[ultimul] ← 0
ultimul ← ultimul -1
          sfârșit  bloc
sfârșit dacă
//sfârsit  secvența  pentru ieșirea unui element

//secvența pentru parcurgerea firului: se afișează elementele din șir și lungimea firului
dacă primul =o
    atunci
            scrie   operatie imposibila:  firul este gol
      altfel
        bloc
            nr ← 0
            pentru i= primul  la ultimul execută
                  bloc
                      scrie  F [i]
                      nr ←  nr + 1
                  sfârșit  bloc
            sfârșit pentru
            scrie  firul contine nr elemente
        sfârșit  bloc
sfârșit dacă
//sfârșit  secvența  pentru  parcurgerea firului de așteptare

sfârșit fir _de_ așteptare_1
```

secvența B

```
dacă primul < ultimul
    atunci
F[primul] ← 0
primul ← primul +1
sfârșit  dacă
```

Pentru simularea aspectului dinamic al firului de așteptare, sugerăm implementarea prel
crărilor specifice printr-un program cu meniu (fig. 24).

```
început  fir _de_ așteptare_2
//se afișează opțiunile din meniu
optiune ← 0
repetă
// se sterge ecranul
// afișare meniu
    repetă
scrie    fir de asteptare –prelucrări specifice
scrie  1 creare
scrie  2 intrarea unui  element nou
scrie  3 iesirea unui element
scrie  4 parcurgere
scrie  5 sfarsit program

scrie  introduceti optiunea (1,2,3,4,5):
citeste optiune
      până când  optiune <=5    // se validează optiunea
   selectează  optiune
optiune=1
            // secvență creare
optiune=2
            // secvență  intrare element nou
optiune=3
            // secvență ieșire element
optiune=4
            // secvență parcurgere
optiune=5
          scrie    sfarsit program
   sfârșit selectează
până când optiune=5

sfârșit fir _de_ așteptare_2
```

Meniu *FIFO*

1. Creare
2. Intrare
3. Servire
4. Parcurgere
5. Exit

Optiune

Figura 24

Printr-o singură rulare a programului, utilizatorul poate să aleagă din meniu, de mai mult
ori, opțiunea corespunzătoare simulării firului de așteptare.

Spre exemplu, pentru simularea intrării primului element urmată de intrarea a încă trei ele
mente și ieșirea a două elemente, utilizatorul poate dirija execuția programului prin următoare
secvență de opțiuni: 1, 4, 2, 4, 2, 4, 2 ,4, 3, 4, 3, 4, 5.

După fiecare opțiune care simul
urgerea (opțiunea 4) prin care se
ontrola dacă programul modelează

TEME

1. Codificați, în limbajul de progra
pentru simularea unui fir de aș
2. Precizați care este efectul următ
a) crearea și parcurgerea firului
b) crearea, intrarea unui nou ele
c) crearea, ieșrea unui element
d) ieșirea unui element din firul
3. Care este efectul eliminării din
așteptare?
4. Compuneți un program nou care
5. Pentru ieșirea unui element
„avansul": secvența A și secvent
a) Care este semnificația mom
b) Cum pot fi utilizate locațiile
6. Realizați un program cu meniu
Mai multe persoane se înscriu
elevii. Se cunosc numărul
Organizatorii doresc să afle num
disponibile mai sunt.
7. Propuneți o situație reală a căre
fire de așteptare.

7.4.2. STIVA

Disciplina structurii dinamice
LIFO (Last Input First Output =
Implementarea într-un limbaj de
ste structură dinamică revine
intrare/ieșire în/din structură, a
disciplina LIFO.

În acest scop, structura va fi
- doi marcatori (indici) de
vârf și *vârf* (fig. 25);
- *capacitatea structurii* (nu
alocate pe care îl vom nota cu *n*)

eptare, sugerăm implementarea prel

Meniu *FIFO*

1. Creare
2. Intrare
3. Servire
4. Parcurgere
5. Exit

Optiune

Figura 24

aleagă din meniu, de mai mult

rmată de intrarea a încă trei ele
ția programului prin următoarea

După fiecare opțiune care simulează dinamica structurii (creare, intrare, ieșire), s-a ales parcurgerea (opțiunea 4) prin care se afișează elementele din fir. În acest mod, utilizatorul poate controla dacă programul modelează corect firul de așteptare.

TEME

1. Codificați, în limbajul de programare studiat, fiecare dintre secvențele pseudocod propuse pentru simularea unui fir de așteptare.
2. Precizați care este efectul următoarei secvențe de opțiuni :
   a) crearea și parcurgerea firului de așteptare;
   b) crearea, intrarea unui nou element și parcurgerea firului de așteptare;
   c) crearea, ieșrea unui element și parcurgerea firului de așteptare;
   d) ieșirea unui element din firul de așteptare.
3. Care este efectul eliminării din program a secvenței de instrucțiuni pentru crearea firului de așteptare?
4. Compuneți un program nou care să nu conțină secvența de creare.
5. Pentru ieșirea unui element din coadă, au fost propuse două secvențe care simulează „avansul": secvența A și secvența B. Urmăriți secvența B și răspundeți la următoarele întrebări:
   a) Care este semnificația momentului *primul = ultimul*?
   b) Cum pot fi utilizate locațiile de memorie eliberate prin ieșirea unui element?
6. Realizați un program cu meniu pentru modelarea următoarei situații:
   Mai multe persoane se înscriu pentru rezervarea de bilete la un spectacol la care au acces doar elevii. Se cunosc numărul de bilete și anul nașterii fiecărei persoane înscrisă pe listă. Organizatorii doresc să afle numărul de persoane înscrise, câți elevi s-au înscris, câte bilete disponibile mai sunt.
7. Propuneți o situație reală a cărei rezolvare cu calculatorul să necesite modelarea datelor prin fire de așteptare.

7.4.2. STIVA

Disciplina structurii dinamice numită *stivă* este de tip LIFO (Last Input First Output =*ultimul intrat, primul ieșit*). Implementarea într-un limbaj de programare a acestui model de structură dinamică revine la controlul operațiilor de intrare/ieșire în/din structură, astfel încât să fie respectată disciplina LIFO.

În acest scop, structura va fi controlată prin:

- doi marcatori (indici) de poziție pe care îi vom numi *bază* și *vârf* (fig. 25);
- *capacitatea structurii* (numărul maxim de elemente alocate pe care îl vom nota cu *n*).

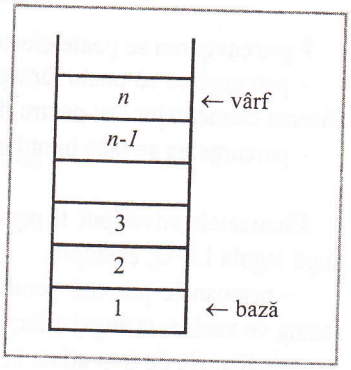


Figura 25

Cazuri particulare

- Dacă stiva nu conține niciun element, stiva este goală (*stivă vidă*).
- Dacă a fost ocupată toată capacitatea structurii, *stiva este plină*.

Prelucrările specifice stivei

▸ **crearea** are sens doar dacă stiva este goală (stivă vidă);

▸ **intrarea** unui element nou:

– un element poate intra în structură doar dacă nu a fost completată capacitatea structurii („mai sunt locuri libere");

– întotdeauna, noul element se așază peste celelalte elemente, în vârful stivei;

– după intrarea unui element nou în stivă, spunem că „stiva crește";

▸ **ieșirea** unui element:

– un element poate ieși din structură, doar dacă stiva nu este vidă (există cel puțin un element);

– întotdeauna, iese din structură elementul aflat în vârful stivei;

– după ieșirea unui element din stivă, spunem că „stiva scade ";

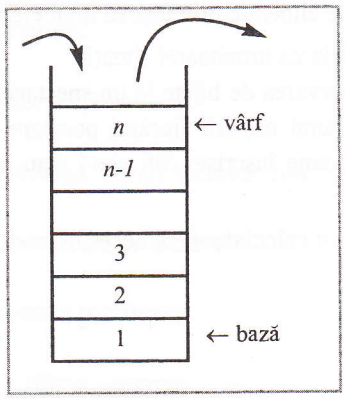


Figura 26

▸ **parcurgerea** se poate efectua doar dacă stiva nu este vidă:

– parcurgerea se poate face pentru determinarea numărului de elemente „stivuite", pentru afișarea elementelor sau pentru determinarea unor proprietăți;

– parcurgerea se face întotdeauna de la *vârf* spre *bază*.

Elementele stivei pot fi persoane, obiecte, procese care așteaptă rezolvarea unei solicitări după regula LIFO; exemple:

– persoanele propuse pentru disponibilizare de la un loc de muncă al cărui manager ține seama de vechimea angajaților;

– vagoanele de tren aflate în probă pe o linie de manevră;

– programele activate de un utilizator *Windows*.

În cele mai frecvente situații, se înregistrează mai multe informații despre elementele aflate în stivă. Fiecare element din stivă este descris ca o structură eterogenă – articol; pentru memorarea elementelor din stivă se va folosi un vector de articole.

Pentru început, vom considera că fiecare element este descris prin numărul său de ordine (vector cu valori întregi, pozitive). În continuare, se prezintă secvența pseudocod a operațiilor necesare implementării stivei.

```
început stiva1
// se lucrează  cu vectorul S pentru care se alocă 100 de elemente
// variabile de lucru:
// n numărul maxim de elemente  din stivă (capacitatea stivei )
// baza  vârf  marcatori de poziție
// i indicele de adresă pentru S
// nr  numărul de elemente din stivă

// secvența de inițializare
// secvența poate fi completată cu validarea lui n față de numărul de elemente alocate (100)

scrie introduceti valoare pentru capacitatea stivei, n=
citește n
pentru i=1 la n execută
S[i] ← 0
sfârșit pentru
//  stiva  este vidă; se consideră că adresarea elementelor din vector începe de la 1
vârf←0
baza←0
// sfârșit secvența de inițializare

// secvența pentru creare
dacă  vârf = 0
atunci
 bloc
        vârf←1
scrie introduceti valoare pentru elementul din varful stivei S[vârf]=
citește S[vârf]
 sfârșit bloc
altfel
     scrie   operatie fara sens:stiva  nu este vida
sfârșit dacă
//sfârșit  secvența creare
```

```
// secvența pentru intrarea unui element nou
vârf ← vârf+1   // stiva crește
dacă vârf < =n
      atunci
            bloc
          scrie introduceti valoare pentru elementul din varful stivei S[vârf]=
           citește S[vârf]
           sfârșit bloc

          altfel
                    scrie   operatie imposibila: stiva este plină
sfârșit dacă
//sfârșit  secvența  pentru intrarea unui element nou
// secvența pentru ieșirea unui element
dacă vârf =0
      atunci
                    scrie   operatie imposibila: stiva  este goala
          altfel
                bloc
 scrie  iese elementul din varful stivei S[varf]
                    // stiva scade
S[varf] ←0
varf ← varf – 1
              sfârșit bloc
sfârșit dacă
//sfârșit  secvența  pentru ieșirea unui element
//secvența pentru parcurgerea stivei:
//se afișează elementele din stivă și înălțimea stivei
dacă vârf =o
      atunci
                    scrie   operatie imposibila: stiva este goala
          altfel
             bloc
                    nr ← 0
                    pentru i= vârf  la baza  execută
                          bloc
                              scrie  S [i]
                                nr ← nr + 1
                           sfârșit  bloc
                     sfârșit pentru
                     scrie  stiva  contine nr elemente
               sfârșit  bloc
sfârșit dacă
//sfârșit  secvența  pentru  parcurgerea stivei
sfârșit stiva1
```

Pentru simularea aspectului dinamic al stivei, sugerăm implementarea prelucrărilor specifice într-un program cu meniu (fig. 27).

```
început   stiva2
//se afișează opțiunile din meniu
optiune←0

repetă
// se șterge ecranul
// afișare meniu
    repetă
scrie   operatii asupra stivei
scrie  1 creare
scrie  2 intrare element nou
scrie  3 iesirea unui element
scrie  4 parcurgere
scrie  5 sfarsit program

scrie introduceti optiunea (1,2,3,4,5):
citește optiune
    până când  optiune <=5    // se validează optiunea

  selectează  optiune
optiune=1
          // secventa creare
optiune=2
          // secventa  intrare element nou
optiune=3
          // secventa iesire element
optiune=4
          // secventa parcurgere
optiune=5
        scrie    sfarsit program
  sfârșit selectează
până când optiune=5
sfârșit stiva2
```

Meniu *LIFO*

1. Creare
2. Intrare
3. Ieșire
4. Parcurgere
5. Exit

Optiune: _

Figura 27

- Printr-o singură rulare a programului, utilizatorul poate să aleagă din meniu, de mai multe ori, opțiunea corespunzătoare simulării stivei.
- Spre exemplu, pentru simularea intrării primului element urmată de intrarea a încă trei elemente și ieșirea a două elemente, utilizatorul poate dirija execuția programului prin următoarea secvență de opțiuni: 1, 4, 2, 4, 2, 4, 2, 4, 3, 4, 3, 4, 5.

• După fiecare opțiune care simulează dinamica structurii (creare, intrare, ieșire) s-a a
parcurgerea (opțiunea 4) prin care se afișează elementele stivei. În acest mod, utilizatorul po
controla dacă programul modelează corect stiva.

TEME

1. Codificați, în limbajul de programare studiat, fiecare dintre secvențele pseudocod propu
pentru simularea stivei.
2. Precizați care este efectul următoarei secvențe de opțiuni:
   a) crearea și parcurgerea stivei;
   b) crearea, intrarea unui nou element și parcurgerea stivei;
   c) crearea, ieșirea unui element și parcurgerea stivei;
   d) ieșirea unui element din stivă.
3. Precizați care este efectul eliminării din program a secvenței de instrucțiuni pentru crea
stivei.
4. Compuneți un program nou care să nu conțină secvența de creare a stivei.
5. Care este rolul marcatorului de poziție *baza* stivei?
6. Compuneți un program nou în care marcatorul *baza* să aibă valoarea *n* (capacitatea stive
7. Realizați un program cu meniu pentru modelarea următoarei situații:
   La biblioteca școlii, cărțile de Informatică sunt atât de solicitate, încât bibliotecara nu le r
pune la loc în raft decât la sfârșitul zilei; în rest, le așază una peste alta pe masă; s-a constata
nici elevii nu selectează cărțile și împrumută de fiecare dată cartea aflată „la vedere" (deasu
celorlalte cărți). Teancul de cărți nu poate depăși *n* exemplare ($n \leq 25$). Se cunoaște autorul, a
editării și prețul fiecărei cărți. Dorim să aflăm, la cerere, numărul cărților, valoarea acestora p
cum și numărul cărților împrumutate într-o zi. (Execuția programului va simula activitatea
restituire și împrumut pe parcursul unei zile.)
8. Propuneți o situație reală a cărei rezolvare cu calculatorul să necesite modelarea datelor p
structuri dinamice de tip stivă.

PROBLEME PROPUSE

1. La un cabinet medical, pacienții intră la consultație în ordinea sosirii. Despre fiecare paci
se cunoaște anul nașterii, genul (M/F), înălțimea și greutatea. Realizați un program care
simuleze înregistrarea pacienților în lista de așteptare, modificarea listei după fiecare con
tație și determinarea următoarelor informații:
   a) numărul pacienților care așteaptă pentru consultație;
   b) numărul pacienților bărbați care așteaptă pentru consultație;
   c) numărul pacienților supraponderali care așteaptă pentru consultație (greutatea ≥ 80 K
înălțimea ≤1,70m).

2. Pentru a rezolva mai repede solicitările unor clienți nemulțumiți de serviciile primite, *Oficiul pentru protecția consumatorilor* a decis să înființeze, pe lângă ghișeul de lucru *G1*, un ghișeu nou, *G2*. Coada formată la ghișeul *G1* este reorganizată astfel: clienții cu număr de ordine *par* trec la ghișeul *G2*.

Realizați un program pentru simularea celor două cozi: *G1* și *G2* pornind de la o coadă inițială, *G1*. Stabiliți singuri datele memorate pentru fiecare client.

3. Realizați un program pentru adunarea a două numere foarte mari. Justificați structura dinamică folosită.

4. Realizați un program pentru afișarea în ordine inversă a unui text introdus de la tastatură (exemplu: textul introdus este *mai mult ca perfectul* textul afișat este *lutcefrep ac tlum iam*). Justificați structura dinamică folosită.

5. Realizați un program pentru transformarea unui număr din baza 10 în baza 2. Justificați structura dinamică folosită.

6. Se consideră formate două cozi C1și C2; în cele două cozi sunt memorate valori numerice. Să se formeze o coadă nouă, C3, prin așezarea elementelor din coada C2 după elementele cozii C1.

Exemplu:
Coada C1 este: 7, 3, 6, 5, 8, 9, 2.
Coada C2 este: 6, 8, 1, 2, 5, 1, 3, 5, 9.
Se obține coada C3: 7, 3, 6, 5, 8, 9, 2, 6, 8, 1, 2, 5, 1, 3, 5, 9.

7. Se consideră formate două stive S1 și S2; în cele două stive sunt memorate valori numerice. Să se formeze o stivă nouă, S3, prin așezarea elementelor din stiva S2 peste elementele stivei S1.

Exemplu:
Stiva S1 este: 7, 3, 6, 5, 8, 9, 2.
Stiva S2 este: 6, 8, 1, 2, 5, 1, 3, 5, 9.
Se obține stiva S3: 6, 8, 1, 2, 5, 1, 3, 5, 9, 7, 3, 6, 5, 8, 9, 2.

8. Realizați un program care să prelucreze o secvență de comenzi pentru un calculator de buzunar; prima comandă executată este prima comandă din secvență. Fiecare comandă are următoarea structură:

operator1 operand operator2

unde: *operator1, operator2* sunt valori numerice reale iar *operand* poate avea valorile *S*-sumă, *D*–diferență *P*–produs, *E*–sfârșitul secvenței de comenzi (*sfârșit prelucrare*).

Exemplu:

Secvența de comenzi	rezultate
3.4 S 3 25.5 D 20.5 12 P 4 E	6.4 5 48 *sfârșit prelucrare*

9. Un șir de caractere conține duplicate în serie (caractere de același fel care se repetă unul după altul). Se dorește rafinarea șirului prin eliminarea duplicatelor în serie. Șirul vid este șir rafinat.

Exemplu:
Șirul inițial este: 912222223444355aa16777777800095.
Șirul final este: 96895.

10. Într-un grup de *n* copii există relații de prietenie; fiecare copil este identificat prin numărul s de ordine în grup; fiecare copil poate fi prieten cu mai mulți copii din grup. Profesorul dirigi dorește să așeze copiii într-o ordine care să pună în evidență relațiile de prietenie, și anum începând cu un copil oarecare *i*, acesta își strigă toți prietenii – în ordinea crescătoare a numă lui; prietenii copilului *i* se așază în rând, unul după altul. Primul copil care urmează în șir d copilul *i* își strigă toți prietenii – în ordinea crescătoare a numărului; aceștia se așază și e rând, unul după altul (dacă un copil a fost strigat înainte, el își păstrează locul în rând).

Exemplul 1: în grup sunt 6 copii.
Relațiile de prietenie dintre copii:

Exemplul 2: în grup sunt 6 copii.
Relațiile de prietenie dintre copii:

Figura 28

a)

	1		1		1
1		1		1	
	1		1	1	
1		1			
	1	1			
1					

Pentru i=2, ordinea copiilor în șir este: 2, 1, 3, 5, 4, 6.

b)

	1				1
1		1			
	1			1	
		1	1		
1					

Pentru i=3, ordinea copiilor în șir este: 3, 2, 5, 1, 4, 6.

Cerințe

a) Specificați structurile necesare organizării datelor din această problemă.
b) Determinați situațiile particulare care pot fi întâlnite în grupul de copii.
c) Formulați exemple numerice care să pună în evidență situațiile particulare determinate
d) Realizați un program care să afișeze aranjarea copiilor din grup în ordinea dorită de p fesorul diriginte.

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Etapa: *Analiză* (identificarea datelor și a prelucrărilor)
Cerințe:
1. Analizați datele și cerințele prezentate în studiul de caz *Compania Eficient* (pag. 3); ju tificați formele de organizare a datelor prin analiza comparativă după eficiența prelucrării ace tora cu calculatorul.
2. Alcătuiți documentația de proiect corespunzătoare acestei etape.

ste identificat prin numărul s
ii din grup. Profesorul dirigin
elațiile de prietenie, și anum
ordinea crescătoare a număr
copil care urmează în șir dup
ărului; aceștia se așază și ei
ăstrează locul în rând).

ul 2: în grup sunt 6 copii.
e de prietenie dintre copii:

			1
1			
		1	
1	1		

=3, ordinea copiilor
e: 3, 2, 5, 1, 4, 6.

problemă.
l de copii.
le particulare determinate.
rup în ordinea dorită de p

EFICIENT

ania Eficient (pag. 3); jus-
ă eficiența prelucrării aces-

ape.

Capitolul 2
SUBPROGRAME

2. UN EXEMPLU DE MODULARIZARE

Modularizarea ca metodă de rezolvare a problemelor s-a dovedit foarte utilă nu numai în programare, ci și în alte domenii ale științei și chiar în viața de zi cu zi.
Rezolvarea acestei „probleme" prin modularizare prezintă următoarele avantaje:

A. ORGANIZAREA FESTIVITĂȚII DE ABSOLVIRE A LICEULUI

Să presupunem că avem de organizat festivitatea de absolvire a liceului.
Pare mult mai greu decât în cazul organizării unei onomastici; de aceea, vom încerca să descompunem „problema" în probleme mai „mici":
(p1) cum vom contacta invitații?
(p2) cum vom organiza partea distractivă?
(p3) cum vom organiza partea culinară?
Problema (**p1**) se poate și ea descompune în alte trei probleme:
(p1.1) lista celor care pot fi anunțați telefonic;
(p1.2) lista celor care pot fi anunțați prin e-mail;
(p1.3) lista celor care pot fi anunțați prin poștă.
Problema (**p2**) se poate descompune în două subprobleme:
(p2.1) stabilirea scenariului, alegerea formațiilor și a soliștilor;
(p2.2) anunțarea formațiilor și a soliștilor invitați.
Problema (**p3**) se poate descompune la rândul ei în următoarele subprobleme:
(p3.1) alcătuirea meniului;
(p3.2) contactarea furnizorilor.

- problemele (**p1.1**), (**p1.2**) , ... , (**p3.2**) sunt mult mai simple și mai ușor de rezolvat decât problema inițială; ele pot fi rezolvate și verificate – independent;
- soluția găsită pentru o anumită subproblemă poate fi folosită și pentru alte subprobleme; de exemplu: problemele (**p2.2**) și (**p2.3**) se pot rezolva la fel ca și problema (**p1**), (fig. 29); acest fapt determină o mare economie de efort și de timp de lucru;

- soluția găsită pentru o anumită subproblemă poate fi adaptată pentru rezolvarea altor p bleme: de exemplu, soluția problemei (**p2.2**) poate fi adaptată și poate conduce rezolvarea problemei (**p3.2**). Și acest fapt determină o mare economie de efort și de ti de lucru;
- având de rezolvat mai multe probleme independente – mai simple – în loc de una sing – mai complicată – soluționarea acestora se poate face independent de către diverse p soane; acest fapt determină, de asemenea, o mare economie de timp de lucru.

Figura 29 ilustrează structurarea și descompunerea problemei de mai sus în subproble (blocurile corespunzătoare problemelor care admit aceeași soluție au fost haşurate identic).

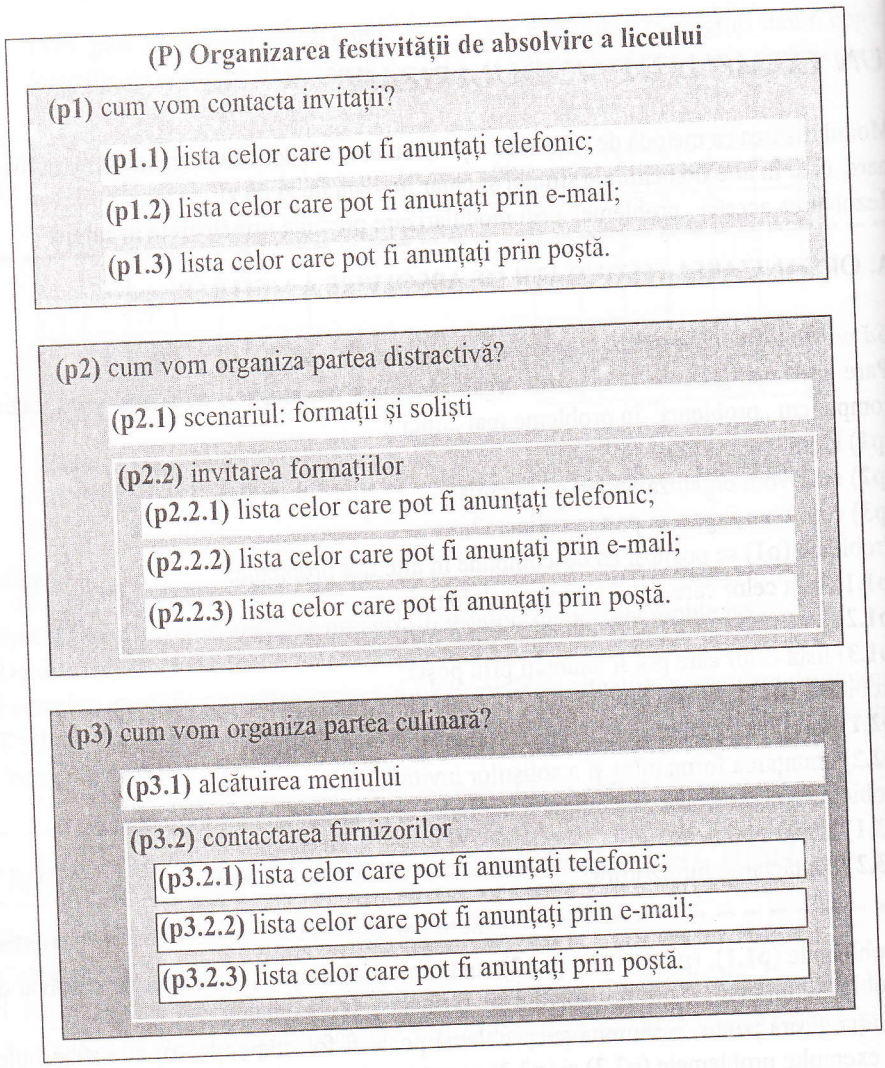


Figura 29. Descompunerea unei probleme în subprobleme

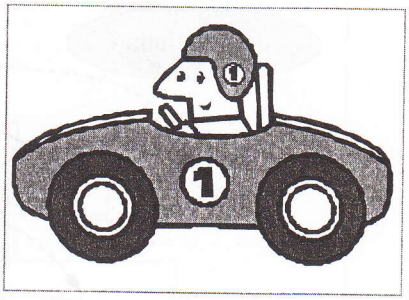
B. CUM FOLOSIM MODULARIZAREA LA NIVELUL UNUI PROGRAM?

Definiție

Un **modul de program** este o unitate de prelucrare care face parte dintr-un program.

Orice program poate fi considerat o funcție în sensul matematic al cuvântului:
- programul primește date de intrare (argumente; de ex. numerele: 1500; 19,44; 13,89; 25);
- programul aplică datelor de intrare un șir de prelucrări (de ex.: înmulțirea, ridicarea la pătrat, împărțirea la 2);
- programul furnizează rezultate (în cazul nostru, valorile: 283.435; 144.699; 468.750).

„Programul" calculează energia cinetică dezvoltată de un corp de masă **m** (aici: un autoturism de 1,5 tone), atunci când viteza sa este egală cu 70, 50, respectiv 90 km/h[1] (adică atunci când circulă în localități, în afara lor, respectiv pe autostrăzi). Cu alte cuvinte, avem de a face cu calcularea energiei cinetice după relația:



$$E_C = \frac{m \cdot v^2}{2} \qquad \text{sau, cu o funcție} \qquad f : R_+ \times R_+ \to R_+ ,\ f(m,v) = \frac{m \cdot v^2}{2}$$

Să simplificăm puțin lucrurile și să presupunem că masa corpului pentru care trebuie calculată energia cinetică este constantă, variind doar viteza. Obținem o funcție cu un singur argument:

$$f_1 : R_+ \to R_+ ,\ f_1(v) = \frac{m}{2} \cdot v^2$$

Dacă examinăm expresia funcției f_1, observăm că aceasta rezultă din compunerea a două funcții: (1) funcția h pentru ridicarea la pătrat:

$$h : R \to R ,\ h(v) = v^2$$

(2) funcția g pentru înmulțirea cu o constantă:

$$g : R \to R ,\ g(w) = k \cdot w ,\ unde\ k = \frac{m}{2}$$

adică:

$$f_1 : R \to R ,\ f_1 = g \circ h ,\ f_1(v) = g(h(v))$$

[1] 70km/h ≈ 19,44 m/s²; 50km/h ≈ 13,89 m/s²; 90km/h ≈ 25 m/s².

În matematică, putem obține funcții oricât de complexe prin **compunerea** funcțiilor e mentare (funcția polinomială, funcția rațională, funcția de ridicare la putere sau de extrage radical etc.).

În programare, putem construi programe oricât de complexe prin tehnica de modulariz adică prin **înlănțuirea logică** a unor „unități elementare" de program: citirea datelor de int prelucrarea lor, afișarea rezultatelor obținute în urma prelucrării datelor. Un modul de prelucr poate rezulta din înlănțuirea logică a mai multor module: calcularea valorii minime dintr-u de valori, calcularea valorii maxime dintr-un set de valori, calcularea mediei aritmeti acestora etc.

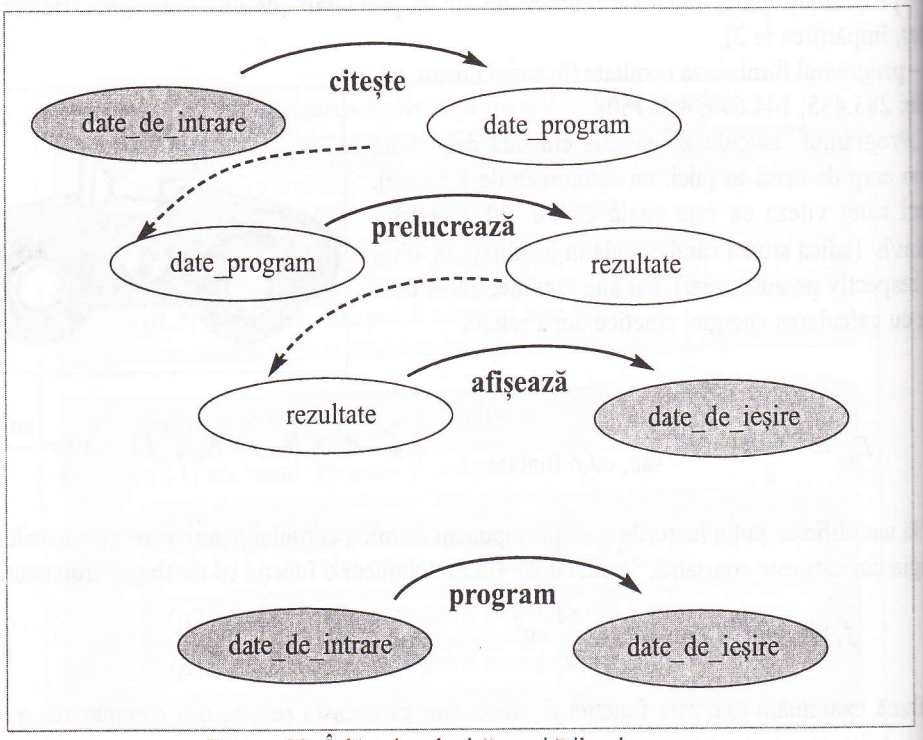


Figura 30. Înlănțuirea logică a unităților de program

Înlănțuirea logică a unităților de program, după **modelul compunerii** funcțiilor matema pune în evidență mulțimile de valori specifice rezolvării problemelor cu calculatorul:

Program (date_de_intrare → date_de_ieșire)
citește (date_de_intrare → date_program)
prelucrează (date_program → rezultate)
afișează (rezultate → date_de_ieșire).

2. MODULARIZAREA PROGRAMELOR

A. TIPURI DE PROBLEME (FACULTATIV)

Rezolvarea unei probleme cu ajutorul unui program de calculator este etapa finală a unui pro- care depinde de gradul în care cunoaștem datele de intrare, tipul rezultatelor, metoda de olvare. Ținând seama de aceste aspecte, problemele cu care suntem zilnic confruntați se pot ifica astfel:

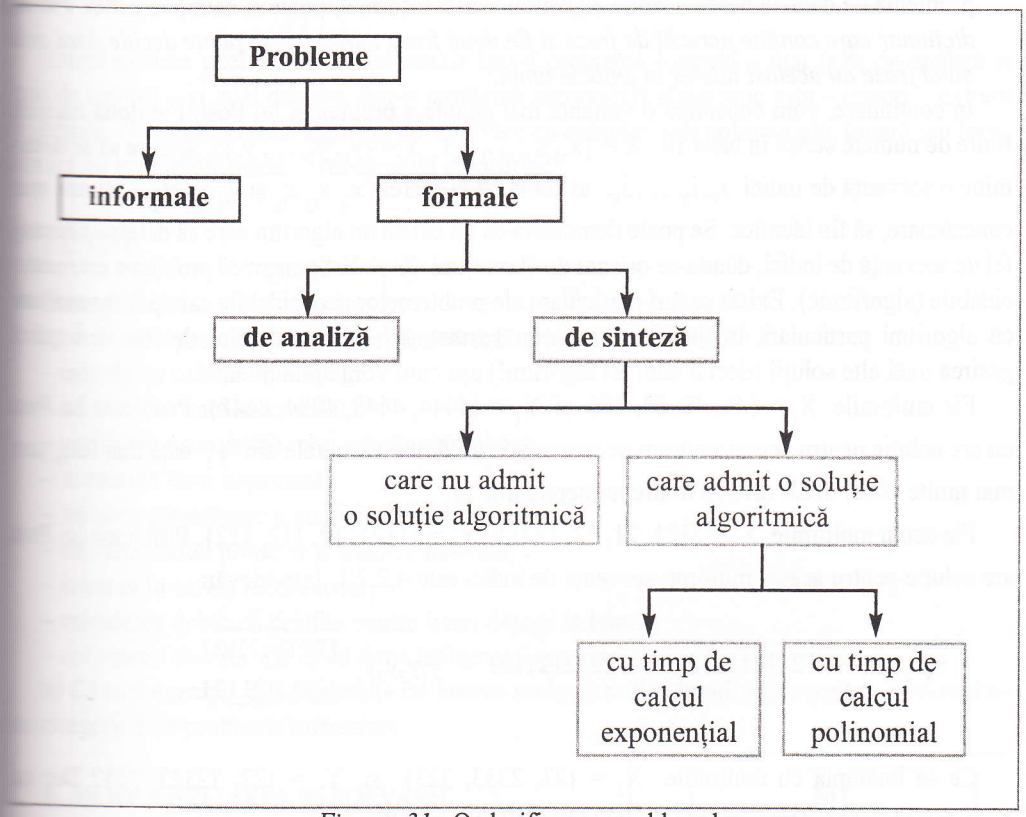


Figura 31. O clasificare a problemelor

Exemple:

- O *problemă informală* este o problemă pentru care nu știm exact de ce date dispunem, nu cunoaștem metoda de rezolvare, nu știm nici măcar cum ar trebui să arate rezultatele. O problemă informală poate fi formulată astfel: *Cum să procedez pentru a fi apreciat de cei din jur?*
- O *problemă formală de analiză* este o problemă pentru care cunoaștem mulțimea datelor inițiale și forma rezultatelor, dar nu cunoaștem metoda de rezolvare; de exemplu: ne aflăm într-un orășel portuar străin, nu cunoaștem nici măcar grafia limbii respective și dorim să ajungem de la hotel în port. Analizând problema, constatăm că nu putem apela la localnici – nici măcar la taximetriști sau hotelieri –, dar, de la lecțiile de geografie, știm că portul se află

în partea sudică a orașului respectiv. Presupunând că avem parte de o zi însorită, ne put totuși orienta să găsim, astfel, drumul.

• O *problemă formală de sinteză* este o problemă pentru care cunoaștem mulțimea date inițiale și metoda de rezolvare. (Foarte multe probleme de matematică, fizică, chimie parte din această categorie.)

• O *problemă de sinteză care nu poate fi rezolvată algoritmic* este mai greu de explicat int itiv. De aceea, vom recurge la prezentarea *problemei corespondenței*[1] *Post*[2]. Aceasta est problemă de decizie nerezolvabilă algoritmic care, informal, poate fi enunțată astfel: *Fie dicționar care conține perechi de fraze și fie două fraze oarecare; se poate decide dacă c două fraze au același înțeles în ambele limbi?*

În continuare, vom considera o variantă mai simplă a problemei lui Post. Fie două mult finite de numere scrise în baza 10, $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$. Se cere să se det mine o secvență de indici $i_1, i_2, \ldots, i_k$, astfel încât numerele $x_{i_1} x_{i_2} x_{i_k}$ și $y_{i_1} y_{i_2} y_{i_k}$, obținute p concatenare, să fie identice. Se poate demonstra că nu există un algoritm care să determine o a fel de secvență de indici, dându-se oricare două mulțimi X și Y. Spunem că problema este ne cidabilă (algoritmic). Există cazuri particulare ale problemelor nedecidabile care pot fi rezolv cu algoritmi particulari, în funcție de exemplul tratat; schimbarea datelor de intrare impl găsirea unei alte soluții (deci a unui alt algoritm!) așa cum vom arăta imediat.

Fie mulțimile $X_1 = \{44, 48, 84, 88\}$ și $Y_1 = \{4444, 4448, 4884, 8448\}$. Problema lui P nu are soluție pentru aceste mulțimi deoarece oricare dintre numerele din Y_1 este mai lung (a mai multe cifre) decât oricare dintre numerele din X_1.

Fie acum mulțimile $X_2 = \{121, 21, 22, 1221\}$, $Y_2 = \{1121, 12, 112, 122\}$. Problema lui P are soluție pentru aceste mulțimi: secvența de indici este 4,2,2,1. Într-adevăr:

$$x_4x_2x_2x_1 = 12212121121 = 12212121121 = y_4y_2y_2y_1$$

$$\overbrace{122121211121}^{x_4\ x_2\ x_2\ x_1}$$

$$\underbrace{122121211121}_{y_4\ y_2\ y_2\ y_1}$$

Ce se întâmplă cu mulțimile $X_3 = \{23, 2333, 323\}$ și $Y_3 = \{22, 32323, 33\}$? Dar c $X_4 = \{22, 2223, 333\}$ și $Y_4 = \{22, 22233, 32\}$? Dar cu...?!

---

[1] Această problemă a fost formulată în anul 1946 și publicată în articolul "A variant of a recursive unsolvable problem", în volumul 52 al Bulletin of the American Mathematical Society. Fiind mai simp decât problema opririi programelor (a se vedea Anexa 1), *problema corespondenței Post* este frecv folosită pentru demonstrarea – prin reducere – a nedecidabilității unor probleme din informatică.

[2] Emil Leon POST (1897 – 1954): matematician de origine poloneză, profesor la City College of Ne York. În teza sa de doctorat, susținută în 1920, Post a demonstrat completitudinea și noncontradicția ca culului cu propoziții – descris de B. Russell și A.N. Whitehead în "Principia Mathematica" – folosind pe tru prima dată metoda tabelelor de adevăr. Este considerat părintele teoriei demonstrației, un precursor lui Kurt Gödel (prin lucrările sale privind mulțimile recursiv enumerabile, gradele de nedecidabilitate) John von Neumann (printr-un model matematic de mașină de calcul foarte asemănător celui descris de v Neumann în celebrul său articol din 1946).

• O *problemă formală de sinteză, care admite o soluție algoritmică dar al cărei ordin de mărime este exponențial,* este problema determinării combinărilor de **n** elemente luate câte **m**, $0 \leq m \leq n$ folosind bine-cunoscuta proprietate a combinărilor.

• O *problemă formală de sinteză, care admite o soluție algoritmică și al cărei ordin de mărime este polinomial* (chiar liniar), este problema căutării secvențiale. Pentru un șir ordonat – dacă utilizăm metoda căutării binare, studiată în clasa a IX-a – complexitatea algoritmului de căutare devine $O(\log_2 n)$.

Transformarea unei probleme informale într-o problemă formală – mai întâi de analiză și apoi de sinteză – și, mai departe, într-o problemă rezolvabilă algoritmic este – uneori – extrem de dificilă. Nici găsirea unui algoritm de rezolvare cu complexitate polinomială, liniară sau logaritmică nu este întotdeauna... rezolvabilă algoritmic!

TEMĂ

Exemplificări

a) În ce categorie de probleme se încadrează următoarele enunțuri:
- calcularea titlului unui aliaj;
- construirea unui aeromodel;
- verificare primalității unui număr natural dat;
- acordarea unui împrumut;
- sortarea crescătoare a unui șir de numere reale;
- calculul sumei primelor ***n*** numere naturale;
- intrarea în cartea recordurilor;
- calcularea dobânzii primite pentru banii depuși la bancă;
- ordonarea elevilor din clasă după înălțime și greutate.

b) Căutați exemple din domeniile de interes școlar sau din situații reale pentru fiecare din-tre categoriile de probleme prezentate.

3. MODULARIZAREA REZOLVĂRII PROBLEMELOR

În practică suntem confruntați, cel mai adesea, fie cu probleme care admit soluții algoritmice și au un grad de complexitate rezonabil, fie numai cu cazuri particulare ale unor probleme nedecidabile, pentru care putem găsi soluții convenabile. În rezolvarea acestor probleme cu ajutorul unui program, folosim modularizarea atunci când căutăm o soluție pentru fiecare dintre următoarele subprobleme (fig. 32):

(M1.) introducerea datelor de intrare (**modulul de introducere a datelor**);

(M2.) prelucrarea datelor (**modulul de prelucrare**);

(M3.) transmiterea datelor de ieșire (**modulul de afișare a rezultatelor**).

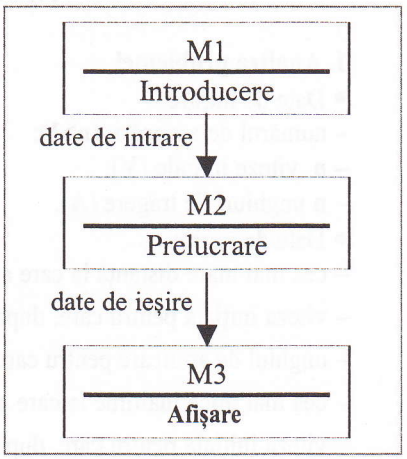


Figura 32. Modularizarea rezolvării unei probleme

Noțiunea de modul este independentă față de limbajul de programare. Un algoritm poate format din mai multe module (subalgoritmi). În continuare, vom folosi termenul de progra fără a implica prin aceasta folosirea unui anumit limbaj de programare.

APLICAȚIE

Problema traiectoriilor

Un tun execută ***n*** *aruncări de proiectile. Pentru fiecare aruncare se cunosc viteza inițial* ***v,*** *și unghiul de tragere* ***α.*** *Trebuie determinate:*

(1) – distanța maximă,

– viteza inițială,

– unghiul de tragere pentru care distanța pe orizontală atinsă de proiectil după ***t*** *secund este maximă;*

(2) – înălțimea minimă,

– viteza inițială,

– unghiul de tragere pentru care înălțimea pe verticală atinsă de proiectil după ***t*** *secund este minimă.*

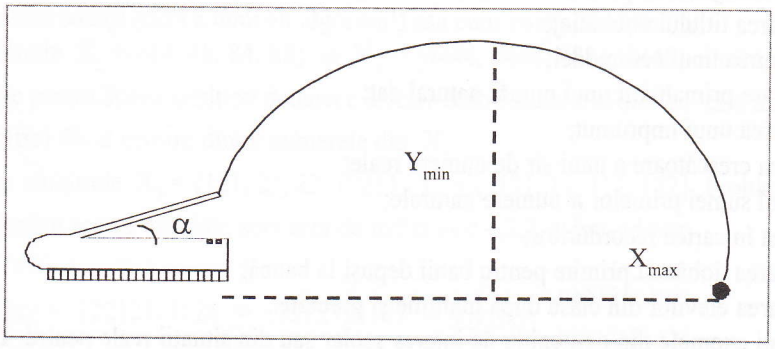


Figura 33. Aruncarea unui proiectil

1. Analiza problemei

• Date de intrare:

– numărul de aruncări (**n>1**);

– **n** viteze inițiale (**V**);

– **n** unghiuri de tragere (**A**).

3	3	5	2	4	6	2	7		

15	30	15	20	45	60	20	75		

• Date de ieșire:

– cea mai mare distanță la care a ajuns un proiectil după **t** secunde (X_{max});

– viteza inițială pentru care, după **t** secunde, proiectilul a atins distanța maximă (V_k);

– unghiul de aruncare pentru care, după **t** secunde, proiectilul a atins distanța maximă (A_k);

– cea mai mică înălțime la care a ajuns un proiectil după **t** secunde (Y_{min});

– viteza inițială pentru care, după **t** secunde, proiectilul a atins înălțimea minimă (V_j);

– unghiul de aruncare pentru care, după **t** secunde, proiectilul a atins înălțimea minimă (A_j)

• Condiții și relații importante:
Se cunoaște **t** = 10 secunde.
Se utilizează formulele de calcu
proiectil:

(1) $x = v \cdot t \cdot \cos\alpha$,

(2) $y = v \cdot t \cdot \sin\alpha - \frac{g \cdot t^2}{2}$

2. Raționamentul problemei
Metoda I: căutăm un algoritm
soluționarea concomitentă a am
Pasul 1. Se citește numărul de
Pasul 2. Se citesc cele **n** per
..., (V_n, A_n).
Pasul 3. Se inițializează consta
Pasul 4. Se calculează distanța
Pasul 5. $X_{max} \leftarrow X_1$, k ←
Pasul 6. Se calculează o nouă
Pasul 7. Dacă $X_i > X_{max}$ atun
Pasul 8. Dacă $Y_i < Y_{min}$ atun
Pasul 9. Se repetă pașii P6, P7
Pasul 10. Se afișează U, distar
unghiul de tragere A_k, viteza V_j

3. Reprezentarea algoritmulu

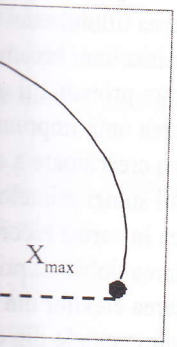
```
început traiectorii1
    variabile n, V, A, X, Y, X
t = 10
g = 9,8
citește n
    pentru i =1 la, n execu
        citește Vi, Ai
    sfârșit pentru
    pentru i =1 la, n execu
        Xi ← Vi * t *
        Yi ← Vi * t *
    sfârșit pentru
    Xmax ← X1
    k ← 1
```

gramare. Un algoritm poate
n folosi termenul de progra
mare.

care se cunosc viteza inițial

ă de proiectil după **t** secun

ă de proiectil după **t** secun



2	7		

20	75		

ade (X_{max});
distanța maximă (V_k);
tins distanța maximă (A_j
ade (Y_{min});
nălțimea minimă (V_j);
ins înălțimea minimă (A_j

• Condiții și relații importante:
Se cunoaște **t** = 10 secunde.
Se utilizează formulele de calcul pentru aflarea distanței (1) și a înălțimii (2) din traiectoria proiectil:

(1) $x = v \cdot t \cdot \cos\alpha$,

(2) $y = v \cdot t \cdot \sin\alpha - \frac{g \cdot t^2}{2}$

2. Raționamentul problemei

Metoda I: căutăm un algoritm cât mai eficient pentru rezolvarea problemei. Aceasta înseamnă soluționarea concomitentă a ambelor cerințe.

Pasul 1. Se citește numărul de aruncări **n.**

Pasul 2. Se citesc cele **n** perechi de viteze inițiale și unghiuri de tragere **(V_1, A_1), (V_2, A_2), ..., (V_n, A_n).**

Pasul 3. Se inițializează constantele **t** = numărul de secunde și **g** = accelerația gravitațională.

Pasul 4. Se calculează distanța X_1 și înălțimea Y_1 conform formulelor (1) și (2).

Pasul 5. $X_{max} \leftarrow X_1$, **k** $\leftarrow$ **1**, $Y_{min} \leftarrow Y_1$, **j** $\leftarrow$ **1.**

Pasul 6. Se calculează o nouă pereche (X_i,Y_i).

Pasul 7. Dacă $X_i > X_{max}$ atunci $X_{max} \leftarrow X_i$, **k** $\leftarrow$ **i.**

Pasul 8. Dacă $Y_i < Y_{min}$ atunci $Y_{min} \leftarrow Y_i$, **j** $\leftarrow$ **i.**

Pasul 9. Se repetă pașii **P6, P7, P8** pentru toate perechile de valori (V_i, A_i).

Pasul 10. Se afișează U, distanța maximă X_{max} și înălțimea minimă Y_{min}, viteza V_k și unghiul de tragere A_k, viteza V_j și unghiul de tragere A_j.

3. Reprezentarea algoritmului

```
început traiectorii1
    variabile n, V, A, X, Y, Xmax, Ymin, k, j, i
t = 10
g = 9,8
citește n
    pentru i =1 la, n execută
            citește Vi, Ai
    sfârșit pentru
    pentru i =1 la, n execută
            Xi ← Vi * t * cos(Ai)
            Yi ← Vi * t * sin(Ai) – g * t * t / 2
    sfârșit pentru
    Xmax ← X1
    k ← 1
```

```
    Ymin ← Y1
    j ← 1
pentru i =2 la, n execută
            dacă Xi > Xmax atunci
                    bloc
                      Xmax ← Xi
                      k ← i
                    sfârșit bloc
            sfârșit dacă
            dacă Yi < Ymin atunci
                    bloc
                      Ymin ← Yi
                      j ← i
                    sfârșit bloc
            sfârșit dacă
    sfârșit pentru
    scrie Xmax, Vk, Ak, Ymin, Vj, Aj
sfârșit traiectorii1
```

Metoda II: rezolvarea prin modularizare

Căutăm să descompunem problema în probleme mai simple, pentru care – eventual – aver soluții, chiar sub formă de subalgoritmi:

(M1) modulul de citire a datelor.

(M1.1) modulul de inițializare a constantelor: **t = 10** și **g = 9,8**;

(M1.2) modulul de citire a variabilelor: numărul de aruncări **n**, cele **n** viteze inițiale $V_1, V_2,..., V_n$ și cele **n** unghiuri de tragere $A_1, A_2,..., A_n$;

(M2) modulul de prelucrare

(M2.1) modulul de calculare a distanțelor $X_1, X_2,..., X_n$ și a înălțimilor $Y_1, Y_2,..., Y_n$ pr aplicarea formulelor (1) și (2);

(M2.2) modulul de calculare a valorii maxime X_{max} dintre $X_1, X_2,..., X_n$ și reținere a inde xului corespunzător **k**;

(M2.3) modulul de calculare a valorii minime Y_{min} dintre $Y_1, Y_2,..., Y_n$ și reținere a inde xului corespunzător **j**;

(M3) modulul de afișare a rezultatelor

afișarea distanței maxime X_{max} , a vitezei inițiale și a unghiului de tragere corespunzător V_k A_k ; afișarea înălțimii minime Y_{min} , a vitezei inițiale și a unghiului de tragere corespunzător V_j A_j.

Raționamentul modularizat este prezentat în figura 34.

(P) **Calcularea distanței maxime și a înălțimii minime atinse de un proiectil din mai multe aruncări**

(M1) citirea datelor de intrare

(M 1.1) modulul de inițializare a constantelor:
t = 10
g = 9,8

(M 1.2) modulul de citire a variabilelor: numărul de aruncări **n**, cele **n** viteze inițiale $V_1, V_2, ..., V_n$ și cele **n** unghiuri de tragere $A_1, A_2, ..., A_n$;

(M2) prelucrarea datelor

(M 2.1) modulul de calculare a distanțelor: $X_1, X_2, ..., X_n$ și a înălțimilor $Y_1, Y_2, ..., Y_n$ prin aplicarea formulelor (1) și (2);

(M 2.2) modulul de calculare a valorii maxime X_{max} dintre $X_1, X_2, ..., X_n$ și reținere a indexului corespunzător **k**;

(M 2.2) modulul de calculare a valorii minime Y_{min} dintre $Y_1, Y_2, ..., Y_n$ și reținere a indexului corespunzător **j**.

(M3) afișarea rezultatelor

distanța maximă X_{max}, viteza V_k și unghiul A_k pentru care a fost atinsă;
înălțimea minimă Y_{min}, viteza V_j și unghiul A_j pentru care a fost atinsă.

Figura 34. Problema traiectoriilor – rezolvare modularizată

3. Reprezentarea algoritmului

```
început traiectorii2
    variabile n, V, A, X, Y, Xmax, Ymin, k, j, i
        t = 10
        g = 9,8
        citește n
        pentru i = 1 la n execută
                citește Vi
        sfârșit pentru
        pentru i ← 1 , n execută
                citește Ai
        sfârșit pentru

        pentru i ← 1, n execută
          Xi ← Vi*t*cos(Ai)
          Yi ← Vi*t*sin(Ai)–9,8*t*t/2
        sfârșit pentru

        Xmax ← X1
        k ← 1
        pentru i ← 2, n execută
          dacă Xi > Xmax atunci
            bloc
              Xmax ← Xi
              k ← i
            sfârșit bloc
          sfârșit dacă
        sfârșit pentru

        Ymin ← Y1
        j ← 1
        pentru i ← 2, n execută
          dacă Yi < Ymin atunci
            bloc
              Ymin ← Yi
              j ← i
            sfârșit bloc
          sfârșit dacă
        sfârșit pentru

    scrie Xmax, Vk, Ak
    scrie Ymin, Vj, Aj
sfârșit traiectorii2
```


Atenție

Programul *traiectorii 1* este
pentru rezolvarea acestei proble
Programul *traiectorii 2*
- poate folosi module deja s
  - modulul pentru deter poziției pe care apare
  - modulul pentru deter poziției pe care apare
- se poate adapta mai ușor;
- prezintă un grad mai mare

Următoarele exemple pun în
Exemplul 1: *modulul de det*
obține din cel de determinare a
valoarea din instrucțiunea de tes

Calcularea minimul

```
început modul minim
variabile Z, VAL, i
VAL ← Z1
pentru i = 2 la n execută
  dacă Zi < VAL atunci VAL←
  sfârșit dacă
sfârșit pentru
returnează VAL
sfârșit minim
```

Exemplul 2: *modulul de deter*
valori se poate obține din modulu
primei instrucțiuni de atribuire:

Calcularea poziți

```
început modul poziție_maxim
variabile Z, VAL, INDEX, i
VAL ← Z1
INDEX ← 1
pentrui = 2 la n execută
dacă Zi > VAL atunci
 bloc
  VAL ← Zi
  INDEX ← i
```

Atenție

Programul *traiectorii 1* este realizat printr-o secvență de operații ce poate fi folosită doar pentru rezolvarea acestei probleme.

Programul *traiectorii 2*

- poate folosi module deja scrise, și anume:
  - modulul pentru determinarea valorii maxime dintr-o secvență de valori și reținere a poziției pe care apare acest maxim,
  - modulul pentru determinarea valorii minime dintr-o secvență de valori și reținere a poziției pe care apare acest minim;
- se poate adapta mai ușor;
- prezintă un grad mai mare de generalitate.

Următoarele exemple pun în evidență flexibilitatea oferită de modularizare.

Exemplul 1: *modulul de determinare a valorii minime dintr-o secvență de valori se poate obține din cel de determinare a valorii maxime (și reciproc) prin modificarea operatorului relațional din instrucțiunea de test:*

Calcularea minimului	Calcularea maximului
început modul minim	început modul maxim
variabile Z, VAL, i	variabile Z, VAL, i
VAL ← Z1	VAL ← Z1
pentru i = 2 la n execută	pentru i = 2 la n execută
dacă Zi < VAL atunci VAL←Zi	dacă Zi > VAL atunci VAL ← Zi
sfârșit dacă	sfârșit dacă
sfârșit pentru	sfârșit pentru
returnează VAL	returnează VAL
sfârșit minim	sfârșit maxim

Exemplul 2: *modulul de determinare a poziției valorii maxime (minime) dintr-o secvență de valori se poate obține din modulul de determinare a valorii maxime (minime) prin adăugarea a unei instrucțiuni de atribuire:*

Calcularea poziției	Calcularea valorii
început modul poziție_maxim	început modul maxim
variabile Z, VAL, INDEX, i	variabile Z, VAL, i
VAL ← Z1	VAL ← Z1
INDEX ← 1	
pentru i = 2 la n execută	pentru i = 2 la n execută
dacă Zi > VAL atunci	dacă Zi > VAL atunci
bloc	
VAL ← Zi	VAL←Zi
INDEX ← i	

sfârșit bloc **sfârșit dacă** **sfârșit pentru** **returnează** VAL, INDEX **sfârșit** poziție_maxim	**sfârșit dacă** **sfârșit pentru** **returnează** VAL **sfârșit** maxim

TEME

Utilizarea modulelor

1. Scrieți un modul de program pentru citirea datelor din problema traiectoriilor, folosind, în l de tablouri unidimensionale, variabile independente. Condiția $n > 1$ este eliminată. Semnal încheierea operației de citire prin:
   a) citirea numărului de aruncări n;
   b) utilizarea unei valori-semafor care să indice oprirea sau continuarea citirii.
2. Scrieți un modul de program care să numere elementele nenule dintr-o secvență de n num date, $n \in N$ fixat.
3. Scrieți un modul de program pentru calcularea mediei aritmetice a unor numere întregi nenul care să poată fi folosit într-un alt modul de program.
4. Se citesc $n \geq 3$ numere întregi strict pozitive mai mari decât 1000. Să se scrie un modul d program care să determine cifra unităților, a sutelor, a miilor și – dacă este cazul – a mil oanelor; cifrele căutate vor fi afișate într-un alt modul de program.
5. Formulați exemple de prelucrări care pot fi organizate în module (subalgoritmi) și utilizate în alte prelucrări (algoritmi).

C. TEHNICI DE MODULARIZARE

Pentru modularizarea programelor putem folosi două tehnici frecvente în proiectare și programare:

(T1.) tehnica *top-down* (de sus în jos, de la complex la simplu; tehnică specifică operației d analiză);

(T2.) tehnica *bottom-up* (de jos în sus, de la simplu la complex; tehnică specifică operație de sinteză).

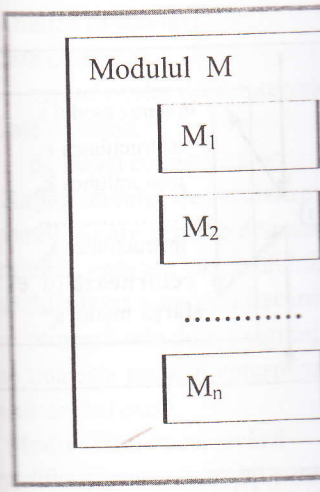
Tehnica *top-down* constă în descompunerea modulelor în submodule până la izolarea celo mai simple unități de lucru (este calea pe care am urmat-o în exemplele prezentate).

Tehnica *bottom-up* procedează exact invers: pornește de la unități de prelucrare elementare și – prin compunerea lor – ajunge la soluția problemei. Acest demers este intuitiv asemănător operației matematice de compunere a funcțiilor.

Două module M_1 și M_2 din cadrul unui program se pot afla unul față de altul în una dintre următoarele relații (ilustrate în figura 35):

(R1) M_2 succede lui M_1; prin urmare, M_1 și M_2 se numesc **module independente**;

(R2) M_2 face parte din M_1; prin urmare M_2 se numește **submodul** (subalgoritm, subprogram) și îi este subordonat modulului (algoritmului, programului) M_1.



Module independente (**R1**)
Figur

În ambele cazuri, M_1 îi trans
numai dacă execuția modulului M_1
Cu alte cuvinte, **modul M_2 se p**
necesare. Aceste date constituie in
pot fi:
– date de intrare, pentru modul
– date de ieșire, pentru modulu

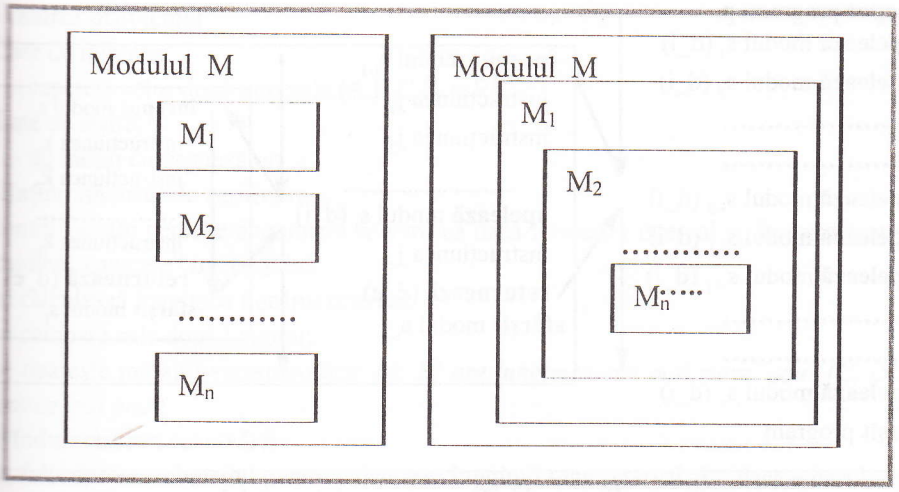
În cazul (**R1**), interfața trebui
lui M_2: datele de ieșire ale modu
intrare pentru acesta.
În cazul (**R2**), interfața trebui
modul (M_2), cât și de la submodu
– elementul de interfață, care
prin cuvântul pseudocod **apelează**

apelează modul(date_d

– elementul de interfață, care
prin cuvântul poseudocod **return**

returnează(date_de

Figura 36 ilustrează modul d
aflate fie în relația (**R1**), fie în r



Module independente (**R1**) Module subordonate (**R2**)

Figura 35. Subordonarea modulelor

În ambele cazuri, M_1 îi transferă controlul lui M_2. Acest transfer **are loc necondiționat,** numai dacă execuția modulului M_1 **s-a încheiat corect**.

Cu alte cuvinte, **modul M_2 se poate executa pentru că modulul M_1 a pregătit toate datele** necesare. Aceste date constituie interfața prin intermediul căreia cele două module comunică și pot fi:

– date de intrare, pentru modului M_2;

– date de ieșire, pentru modului M_1.

În cazul (**R1**), interfața trebuie să asigure numai comunicarea de la modulul M_1 la modulul M_2: datele de ieșire ale modulului M_1 trebuie transmise modulului M_2 și devin date de intrare pentru acesta.

În cazul (**R2**), interfața trebuie să asigure atât comunicarea de la modulul (M_1) la submodulul (M_2), cât și de la submodulul **M2** către modulul **M1**:

– elementul de interfață, care asigură comunicarea de la modul spre submodul, se introduce prin cuvântul pseudocod **apelează** urmat de numele submodulului și mulțimea datelor de intrare;

apelează modul(date_de_intrare)

– elementul de interfață, care asigură comunicarea de la submodul spre modul, se introduce prin cuvântul poseudocod **returnează,** urmat de mulțimea datelor de ieșire.

returnează(date_de_ieșire)

Figura 36 ilustrează modul de comunicare într-un program compus din mai multe module, aflate fie în relația (**R1**), fie în relația (**R2**).

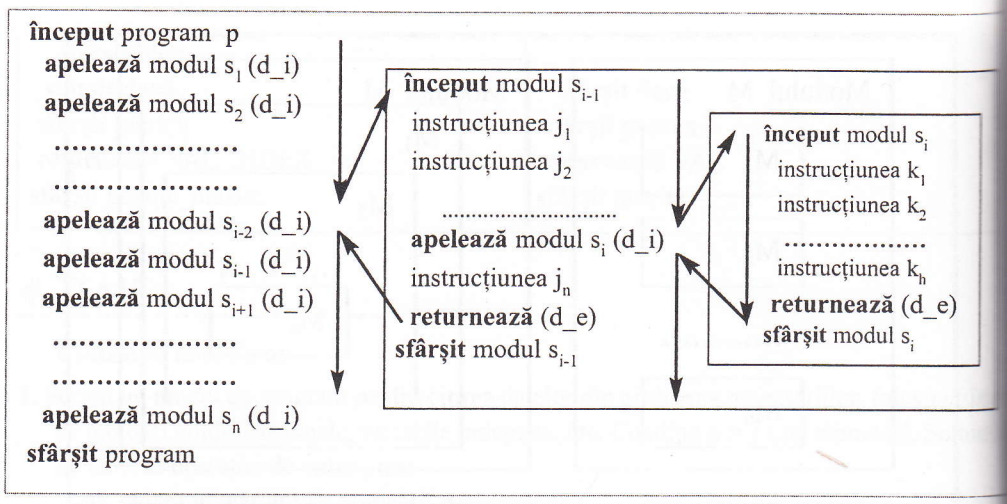


Figura 36. Comunicarea între module

TEME

Comunicarea între module

1. Proiectați un modul de program care să numere zilele care au trecut de la începutul anului până în ziua curentă. Ce date de intrare primește modulul? Ce date de ieșire returnează modulul?

2. Se citesc ***n*** perechi de numere întregi. Proiectați un program modularizat pentru:

a) determinarea mediei aritmetice a fiecărei perechi de numere;

b) determinarea perechilor de numere pentru care media aritmetică are valoarea maximă.

Câte module poate avea programul? De câte ori poate fi apelat fiecare modul?

D. IMPLEMENTAREA MODULARIZĂRII. STIVA SISTEM

Fiecare apel de modul determină o întrerupere a prelucrărilor din modulul apelant. Pentru reluarea acestora după revenirea în modul, adresa operației la care s-a făcut întreruperea este reținută într-o zonă specială de memorie, la dispoziția procesorului, numită **stiva sistem**. Pe același nivel al stivei sistem sunt memorate și datele de intrare în modul. Cu fiecare apel/întrerupere, stiva sistem crește; la revenirea din modulul apelat, stiva sistem scade.

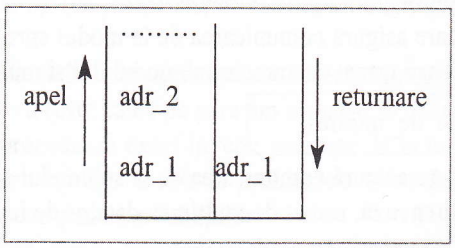


Figura 37. Stiva sistem

Exemplu: Intervalul de lungime maximă.

Fie două intervale de numere reale [A, B] și [C, D]. Se cere să se determine intervalul de lungime maximă.

1. Analiza problemei

- Date de intrare:

  capetele celor două intervale (A,B,C,D ∈ R).

- Date de ieșire:

  un mesaj corespunzător.

2. Raționamentul de rezolvare

- pentru fiecare pereche de numere se verifică dacă formează interval și, în caz negativ, se schimbă „capetele" intervalului;
- se calculează lungimea fiecărui interval;
- se compară cele două lungimi;
- se tipărește mesajul corespunzător: *[A, B] are lungimea cea mai mare sau [C, D] are lungimea ce mai mare.*

3. Modularizarea rezolvării

- se folosește un submodul pentru calcularea lungimii unui interval și – dacă e cazul – pentru interschimbarea „capetelor" intervalului.

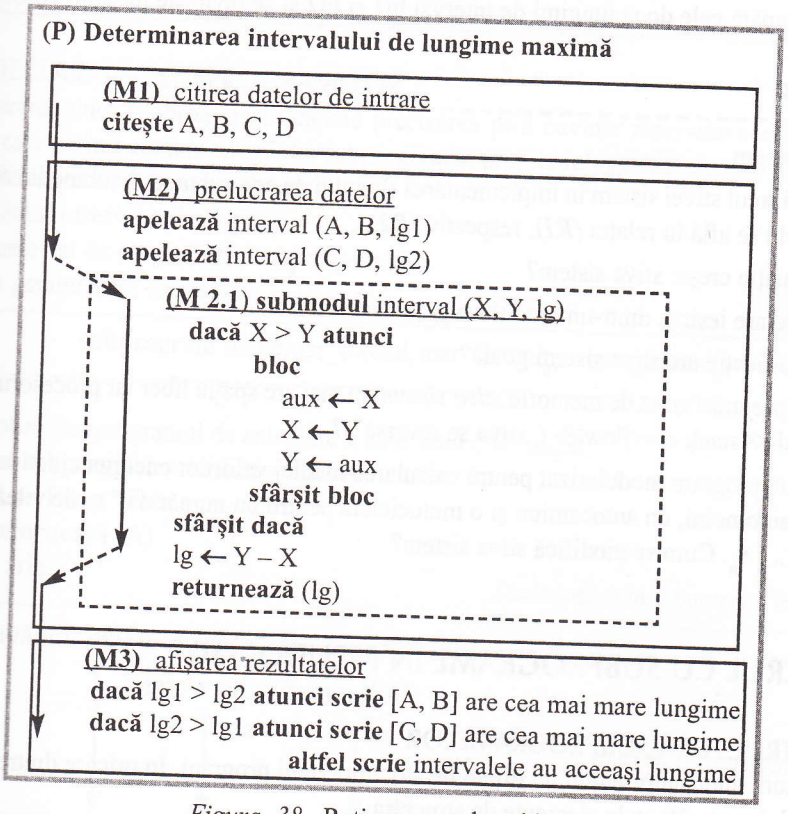


Figura 38. Raționamentul problemei

4. Descrierea fluxului de control

- se intră în modulul de citire a datelor de intrare (**M1**) ⇒

– se citesc cele patru numerele reale **A, B, C, D**;

- se intră în modulul de prelucrare (**M2**) ⇒
  - se apelează submodulul (**M 2.1**) pentru perechea **A** și **B** ⇒
  - se memorează adresa de întrerupere în stiva sistem;
  - adresele valorilor **A** și **B** sunt depuse în stiva sistem;
  - se execută submodulul (**M 2.1**) pentru perechea **A** și **B** ⇒
  - lungimea intervalului [A, B], este transmisă modulului (**M2**),
- stiva sistem scade;
  - se apelează submodulul (**M 2.1**) pentru perechea **C** și **D** ⇒
  - se memorează adresa de întrerupere în stiva sistem,
  - adresele valorilor **C** și **D** sunt depuse în stiva sistem;
  - se execută submodulul (**M 2.1**) pentru perechea **C** și **D** ⇒
- lungimea intervalului [C, D], este transmisă modulului (**M2**),
- se revine în modulul **(M2)**
- stiva sistem scade;
- se intră în modulul de afișare a rezultatelor (**M3**) ⇒
- se compară cele două lungimi de interval **lg1** și **lg2** și se tipărește mesajul corespunzător

TEME

Stiva sistem

1. Explicați rolul stivei sistem în implementarea fluxului de control modul-submodul atunci când modulele se află în relația *(R1)*, respectiv *(R2)*.
2. În ce situație crește stiva sistem?
3. Ce efecte are ieșirea dintr-un modul asupra stivei sistem?
4. Ce semnificație are stiva sistem goală?
5. În ce împrejurări zona de memorie ***stiva sistem*** nu mai are spațiu liber iar procesorul transmite mesajul <<stack overflow>> („stiva se revarsă")?
6. Scrieți un program modularizat pentru calcularea mediei valorilor energiei cinetice dezvoltate de un automobil, un autocamion și o motocicletă pentru un număr dat **n** de viteze de rulare $v_1, v_2, \ldots, v_n$. Cum se modifică stiva sistem?

3. LUCRUL CU SUBPROGRAME ÎN PSEUDOCOD

A. STRUCTURA SUBPROGRAMELOR

Structura unui subprogram nu diferă de structura unui program. În oricare dintre aceste module, întâlnim următoarele elemente de structură:

- un antet;
- o secțiune (opțională) de declarații (constante, variabile, alte subprograme);
- o secțiune de instrucțiuni executabile (simple sau structurate).

B. DEFINIREA SUBPROGRA

În cele mai multe dintre limba
secțiunii de instrucțiuni a progra
Definirea unui subprogram înseam
- alegerea unui nume;
- stabilirea datelor de intrare și
- scrierea instrucțiunilor prin
obținerii datelor de ieșire din subpr

Atenție

Mulțimea variabilelor care
programului formează lista par
Acești parametri apar în inte
ma a sublinia similaritatea dintr
prin punct și virgulă.

C. DECLARAREA SUBPROG

Declararea unui subprogram în
interfață care permit comunicarea î
descrise în **antetul** subprogramului
- cuvântul rezervat **subprogra**
- numele dat de programator su
- lista parametrilor formali.

subprogram num

Exemplu: Subprogramul de ca
început subprogram arie (Lun
A ← Lung * Lat
returnează (A)
sfârșit arie

cuvântul rezervat pentru
numele sub
da

început subprogram arie
A ← Lung * Lat
returnează (A)
sfârșit arie

B. DEFINIREA SUBPROGRAMELOR

În cele mai multe dintre limbajele de programare, subprogramele sunt plasate „înaintea" secțiunii de instrucțiuni a programului propriu-zis (numit, de obicei, **program principal**). Definirea unui subprogram înseamnă:

- alegerea unui nume;
- stabilirea datelor de intrare și a datelor de ieșire proprii subprogramului;
- scrierea instrucțiunilor prin care se realizează prelucrarea datelor de intrare în vederea obținerii datelor de ieșire din subprogram.

> **Atenție**
>
> **Mulțimea variabilelor care constituie datele de intrare și datele de ieșire ale subprogramului formează lista parametrilor formali ai subprogramului.**
>
> **Acești parametri apar în interiorul unei perechi de paranteze rotunde (tocmai pentru a sublinia similaritatea dintre subprograme și funcțiile matematice) și sunt separați prin punct și virgulă.**

C. DECLARAREA SUBPROGRAMELOR

Declararea unui subprogram înseamnă precizarea prin cuvinte rezervate a elementelor de interfață care permit comunicarea între subprogram și programul apelant. Aceste elemente sunt descrise în **antetul** subprogramului prin:

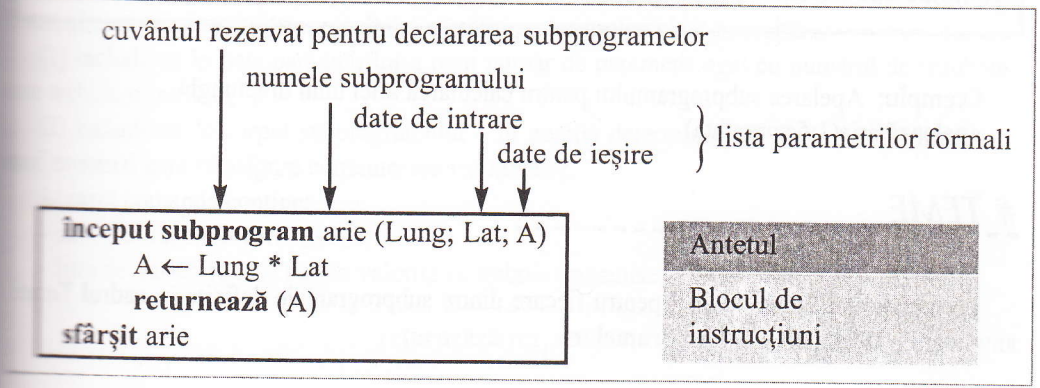
- cuvântul rezervat **subprogram**;
- numele dat de programator subprogramului;
- lista parametrilor formali.

subprogram nume(par_formal$_1$;par_formal$_2$;...;par_formal$_n$)

Exemplu: Subprogramul de calculare a ariei unui dreptunghi:

```
început subprogram arie (Lung; Lat; A)
    A ← Lung * Lat
    returnează (A)
sfârșit arie
```

Figura 39.
Declararea unui subprogram în pseudocod



TEME

Declararea subprogramelor

Declarați câte un subprogram pentru rezolvarea următoarelor cerințe:

a) calcularea ariei și volumului unei sfere de rază dată;

b) calcularea perimetrului și ariei unui triunghi, cunoscându-i laturile;

c) calcularea greutății unui obiect de masă dată;

d) calcularea densității unui lichid, cunoscându-i masa și volumul.

Explicați semnificația parametrilor pentru fiecare dintre subprograme.

D. APELAREA SUBPROGRAMELOR

Orice program care apelează un subprogram se numește **program apelant**, deoarece nu numai programul principal admite subprograme, ci și un subprogram oarecare poate avea, la rândul său, subprograme (figurile 35 și 36).

Prin **apelarea unui subprogram S de către un program apelant P** înțelegem o comandă pe care programul apelant o trimite subprogramului și prin care îi cere acestuia să execute prelucrarea din blocul său de instrucțiuni. Prin această comandă, programul apelant **P** trebuie:

(1) să-i furnizeze subprogramului **S** datele de intrare necesare prelucrării;

(2) să-i indice datele de ieșire în care subprogramul **S** trebuie să-i furnizeze rezultatul – sau rezultatele – prelucrărilor sale.

Toate aceste informații sunt transmise subprogramului cu ajutorul listei de parametri din apelul subprogramului. Acești parametri se numesc **parametri actuali**. Elementele comenzii de apel sunt:

– cuvântul rezervat **apelează**;

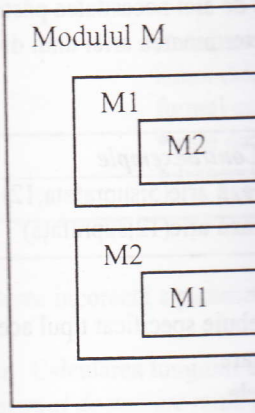
– numele subprogramului;

– lista parametrilor actuali.

apelează nume(par_actual$_1$;par_actual$_2$;…;par_actual$_n$)

Atenție

Parametrii actuali din comanda de apel trebuie să corespundă – ca tip, număr și ordine de enumerare – parametrilor formali din antetul subprogramului.



Exemplu: Apelarea subprogramului pentru calcularea ariei unui dreptunghi

apelează arie(1;5;suprafața)

TEME

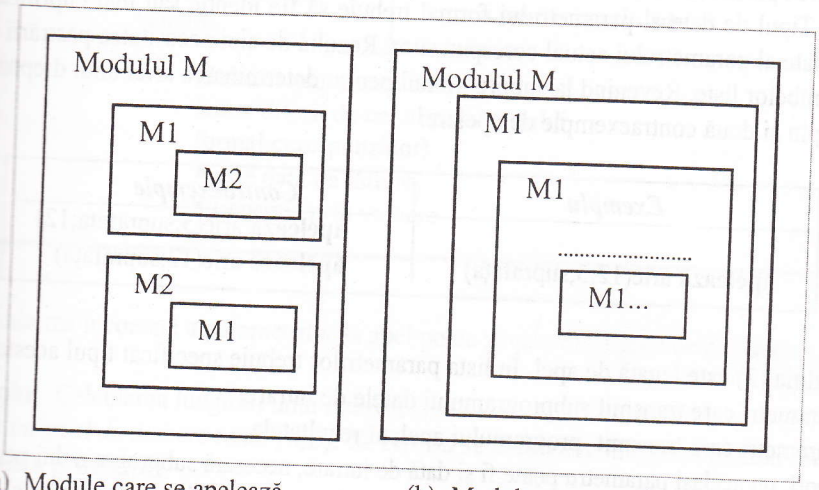
Apelarea subprogramelor

Formulați comenzile de apel pentru fiecare dintre subprogramele definite în cadrul Temei anterioare – **Declararea subprogramelor**.

Atenție

De ce este nevoie de două liste de parametri: o listă de parametri formali și o listă de parametri actuali? Dacă ne-am limita la o singură listă (lista parametrilor formali), aceasta ar trebui să apară – evident! – în antetul subprogramului în momentul definirii sale, iar la apel, toate programele apelante ar trebui să cunoască și să utilizeze exact această listă. Existența celei de-a doua liste, lista parametrilor actuali – cu respectarea convențiilor enumerate mai sus – asigură independența subprogamului apelat față de oricare dintre programele apelante.

Figura 40 ilustrează două cazuri particulare de apelare a subprogramelor.



(a) Module care se apelează unul pe celălalt

(b) Modul care se apelează pe el însuși

Figura 40. Apelarea subprogramelor – cazuri particulare

E. RETURNAREA VALORILOR CĂTRE PROGRAMUL APELANT

Un subprogram trebuie să producă un rezultat – prin prelucrările efectuate asupra datelor de intrare. De fapt, un program apelează subprogramul atunci când are nevoie de acel rezultat. Comunicarea rezultatului (sau rezultatelor) către programul apelant necesită:

(1) includerea în lista parametrilor a unui număr de parametri egal cu numărul de rezultate care trebuie transmise;

(2) includerea în corpul subprogramului – în poziția determinată de fluxul controlului – a unei comenzi care să asigure comunicarea rezultatelor.

Această comandă conține:

– cuvântul rezervat **returnează**,

– lista de variabile care conțin valorile ce trebuie transmise.

returnează(rez_1,rez_2,…,rez_k)

Exemplu: Returnarea valorii ariei dreptunghiului către programul apelant
returnează (A)

F. TRANSFERUL PARAMETRILOR LA APEL

Partea cea mai importantă din instrucțiunea de apelare a subprogramelor este lista parametrilor actuali. În scrierea ei, trebuie să pornim de la lista parametrilor formali și să respectăm:

(1) corespondența dintre parametrii formali și parametrii actuali ca tip, număr și ordine;

(2) modul de transmitere a parametrilor: prin valoare sau prin adresă.

Condiția (1) este legată de forma apelului. În apel apar variabile de program (simple sau structurate), caracterizate aici prin *nume* și *tip*. Un parametru actual poate avea același nume ca parametrul formal corespunzător sau poate avea un nume diferit. În general, este recomandabilă redenumirea parametrilor formali în lista parametrilor actuali doar acolo unde – altfel – s-ar crea confuzii. Tipul de date al parametrului formal trebuie să fie identic sau mai cuprinzător decât tipul de date al parametrului actual corespunzător. Rezultă de aici necesitatea păstrării ordinii la nivelul ambelor liste. Revenind la subprogramul pentru determinarea ariei unui dreptunghi, iată un exemplu și două contraexemple de apelare:

Exemplu	*Contraexemple*
	apelează arie(5;suprafața;12)
apelează arie(12;5;suprafața)	**apelează** arie(12;suprafața)

Condiția (2) este legată de apel. În lista parametrilor trebuie specificat tipul acestora:

– parametri care transmit subprogramului datele de intrare;

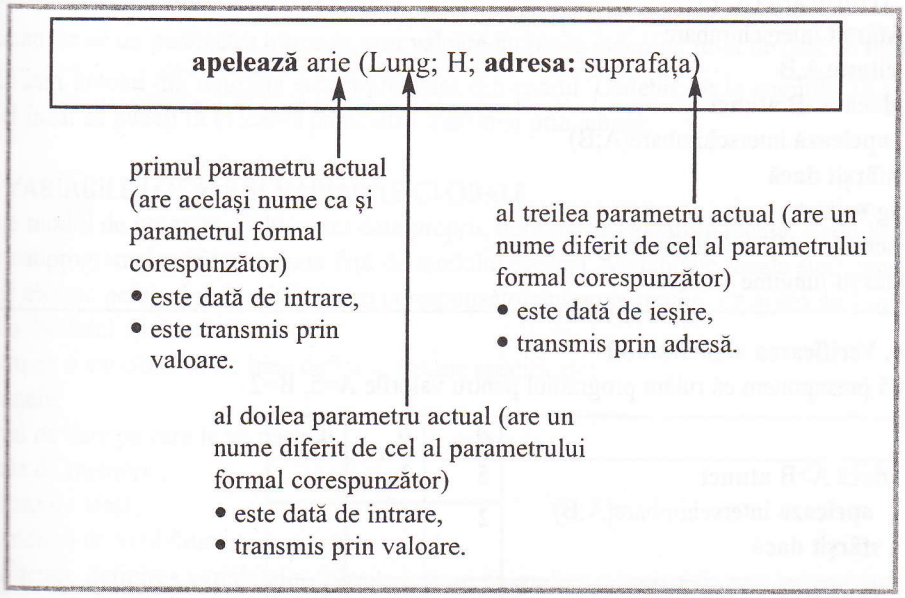
– parametri care transmit programului apelant rezultatele.

Uneori, un același parametru poate fi și dată de intrare, necesară subprogramului pentru efectuarea prelucrărilor, și dată de ieșire, prin care să transmită programului apelant rezultatul prelucrării.

Informația depusă într-un parametru-dată de ieșire sau într-un parametru-dată de intrare-ieșire trebuie să fie accesibilă atât pentru subprogram, cât și pentru programul apelant (care o poate prelua în vederea unor noi prelucrări). Din această cauză, este necesar să comunicăm compilatorului că trebuie să aloce **aceeași locație de memorie** atât variabilei-parametru formal, cât și variabilei-parametru actual – chiar dacă ele au nume diferite. Pentru ca zona de memorie să fie „vizibilă" (accesibilă), subprogramul trebuie să cunoască adresa acestei zone. În acest caz parametrul va fi transmis **prin adresă**, lucru semnalat – numai în lista parametrilor actuali – printr-un cuvânt rezervat, **adresă**, care precede numele parametrului în listă. Spunem că **transmitem** acel **parametru** (dată de ieșire sau dată de intrare-ieșire) **prin adresă** (sau prin referință).

Un parametru care este dată de intrare pentru subprogram trebuie să fie accesibil acestuia (este principala modalitate de primire a datelor de către subprogram), dar trebuie să fie protejat împotriva unor modificări prin prelucrările din subprogram. Din această cauză, este necesar să comunicăm compilatorului că trebuie să aloce acelei variabile **două locații de memorie**: una accesibilă programului – definită în segmentul de date – pe care subprogramul nu o poate modifica, și una accesibilă subprogramului – definită pe un nivel de stivă – pe care subprogramul o poate modifica oricum. Spunem că **transmitem** acel **parametru** (dată de intrare) **prin valoare**

Exemplu: Transmiterea parametrilor la apelul subprogramului pentru determinarea ariei unui dreptunghi:



Transmiterea incorectă a parametrilor la apel poate genera erori greu de depistat.

Exemplu: Calcularea lungimii unui interval

Fie un interval de numere reale [A,B]. Se cere să se determine lungimea acestui interval.

1. **Analiza problemei**

• Date de intrare:
capetele intervalului (A,B ∈ R).

• Date de ieșire:
lungimea intervalului (lg ∈ R).

2. **Raționamentul problemei**

• se verifică dacă numerele A și B formează interval și, dacă este cazul, se interschimbă aceste valori;

• se calculează lungimea intervalului;

• se folosește un submodul pentru interschimbarea "capetelor" intervalului.

3. **Reprezentarea algoritmului**

```
început program lungime_interval
variabile A,B,lg
început subprogram interschimbare(adresă:X;adresă:Y)
variabile aux
aux ← X
```

```
    X ← Y
    Y ← aux
    returnează (X;Y)
sfârșit interschimbare
citește A,B
dacă A>B atunci
 apelează interschimbare(A;B)
sfârșit dacă
lg ← B-A
scrie [A, B] are lungimea lg
sfârșit lungime_interval
```

4. Verificarea algoritmului

Să presupunem că rulăm programul pentru valorile **A**=5, **B**=2

	A	*B*	*lg*
dacă A>B **atunci**	**5**	**2**	
apelează interschimbare(A;B) **sfârșit dacă**	**2**	**5**	
lg ← B-A			**3**

Rezultatele obținute sunt corecte deoarece am transmis parametrii subprogramului *interschimbare* prin adresă (ambii sunt date de intrare-ieșire).

Să presupunem că transmitem acești parametri prin valoare, adică antetul subprogramului este:

început subprogram interschimbare(X;Y)

În acest caz, rezultatul obținut va fi –3 deoarece programul apelant nu "vede" zona de memorie în care subprogramul a efectuat interschimbarea și calculează lg ← 2 – 5.

	Stiva sistem		***Segmentul de date***		
	A	*B*	*A*	*B*	*lg*
citește A,B			**5**	**2**	
dacă A>B **atunci**	**5**	**2**			
apelează interschimbare(A;B) **sfârșit dacă**	**2**	**5**			
lg ← B-A					**– 3**
scrie [A, B] are lungimea lg					

TEME

Transmiterea parametrilor

1. Explicați de ce un parametru tr
2. Modificați antetul din definiți astfel încât să puneți în evide

C. VARIABILE LOCALE ȘI

Orice modul de program prel conferă subprogramelor indepen memorie alocate pe nivelul de sti zilele din modulul apelant.

Pentru ca o variabilă să fie bi

- numele;
- tipul de date pe care le va p
- zona de memorie;
- durata de viață;
- domeniul de vizibilitate.

Până acum, definirea variabil la tipului lor.

Prin **zona de memorie** a unei memorată (unde se află locat dintre următoarele zone de m

- segmentul de date;
- segmentul de stivă;
- *heap*-ul;

sau într-un registru dedicat al

Prin **durata de viață** a unei v în care variabila dispune de compilator (la sfârșitul interva diferită, deci ultima valoare a v memoriu, varibilele se clasifică în:

- **variabile statice**: locațiile c pentru perioada execuției progr
- **variabile dinamice**: locați alocată în *heap*).

Prin **domeniul de vizibilitat** instrucțiuni) în care variabila po definite, variabilele pot fi:

- **variabile locale** (definite în
- **variabile globale** (definite

TEME

Transmiterea parametrilor prin adresă

1. Explicați de ce un parametru transmis prin valoare nu poate îndeplini rolul de dată de ieșire.
2. Modificați antetul din definiția subprogramelor din cadrul **Temelor** de la paginile 78 și 82 astfel încât să puneți în evidență parametrii transmiși prin adresă.

C. VARIABILE LOCALE ȘI VARIABILE GLOBALE

Orice modul de program prelucrează date proprii, definite ca **variabile locale**. Acest aspect conferă subprogramelor independență față de modulul apelant. Variabilele locale sunt zone de memorie alocate pe nivelul de stivă sistem corespunzător subprogramului; de aceea nu sunt vizibile din modulul apelant.

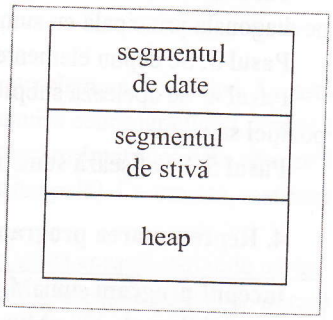
Pentru ca o variabilă să fie bine definită, trebuie specificate:

- numele;
- tipul de date pe care le va păstra;
- zona de memorie;
- durata de viață;
- domeniul de vizibilitate.

Până acum, definirea variabilelor folosite într-un program s-a limitat la precizarea numelui și a tipului lor.

Prin **zona de memorie** a unei variabile înțelegem tipul de memorie internă în care variabila este memorată (unde se află locația atribuită ei). Variabilele oricărui program pot fi memorate în una dintre următoarele zone de memorie internă:

- segmentul de date;
- segmentul de stivă;
- *heap*-ul;
- într-un registru dedicat al microprocesorului.



Prin **durata de viață** a unei variabile înțelegem intervalul de timp în care variabila dispune de locația de memorie atribuită ei de compilator (la sfârșitul intervalului, locația de memorie va fi eliberată, deci ultima valoare a variabilei se pierde). După acest criteriu, variabilele se clasifică în:

- **variabile statice**: locațiile de memorie se alocă în momentul compilării și sunt disponibile pe toată perioada execuției programului (alocare statică în segmentul de date);
- **variabile dinamice**: locațiile de memorie se alocă în momentul execuției (alocare dinamică în *heap*).

Prin **domeniul de vizibilitate** al unei variabile înțelegem segmentul de program (blocul de instrucțiuni) în care variabila poate fi utilizată. În raport cu segmentul de program în care sunt definite, variabilele pot fi:

- **variabile locale** (definite în subprogramul apelat);
- **variabile globale** (definite în programul apelant).

Exemplu: Variabile locale și globale

Fie o matrice pătratică cu elemente numere reale; se cere:

a) să se înlocuiască fiecare element de pe diagonala principală a matricei cu suma elementelor de pe linia corespunzătoare (inclusiv elementul de pe diagonală);

b) să se afișeze indexul liniei pentru care suma elementelor este minimă, precum și valoarea acestei sume.

1. Analiza problemei

- Date de intrare:

– numărul de linii și coloane (**n**),

– matricea (**A**).

- Date de ieșire:

– valoarea celei mai mici sume (Val),

– indexul liniei corespunzătoare acestei sume (pMin).

2. Modularizarea rezolvării

Vom folosi:

– un subprogram pentru calcularea sumei elementelor de pe liniile matricei și depunerea acestor sume pe diagonala principală;

– un subprogram de determinare a minimului unui vector cu **k** elemente și a poziției sale.

3. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: **n, A.**

Pasul 2. Se apelează subprogramul de calculare a sumei pe linii și înlocuire a elementului de pe diagonala principală cu suma respectivă.

Pasul 3. Se depun elementele de pe diagonala principală a matricei într-un vector.

Pasul 4. Se apelează subprogramul de determinare a elementului minim dintr-un vector și a poziției sale.

Pasul 5. Se afișează suma minimă și indexul liniei pe care se află aceasta.

4. Reprezentarea programului cu subprograme

```
început program sumaMinimă
  variabile n, A, Val,pMin,V,i   //        globale
  început subprogram diag(adresa:mat)
    variabile suma,i,j            //        locale
    pentru i=1 la n execută
       suma ← mat(i,1)
       pentru j=2 la n execută
         suma ← suma+mat(i,j)
       sfârșit pentru
       mat(i,i) ← suma
    sfârșit pentru
    returnează mat
```

```
sfârșit diag
început subprogram pozMinim(V;k; adresa:pMin)
 variabile i                    //      locale
 Val ← V(1)
 pMin ← 1
 pentru i=2 la k execută
    dacă V(i) < Val
    atunci
       Val ← V(i)
       pMin ← i
    sfârșit dacă
 sfârșit pentru
 returnează pMin
sfârșit pozMinim
citește n,A
apelează diag(A)
pentru i=1 la n execută
 V(i) ← A(i,i)
sfârșit pentru
apelează pozMinim (V;n;pMin)
scrie Val,pMin
sfârșit sumaMinimă
```

5. Analiza programului

Examinând acest program, observăm:

- lista de parametri a subprogramului *diag* constă dintr-un singur element: o matrice. Aceasta reprezintă atât datele de intrare (elementele matricei trebuie transmise subprogramului pentru a permite calcularea sumelor), cât și datele de ieșire pentru subprogram (matricea este returnată după ce a fost modificată prin depunerea sumelor pe diagonala principală). Ca urmare, matricea este transmisă prin adresă nu prin valoare;
- n = numărul de linii din matrice este o variabilă globală; de aceea această variabilă nu trebuie transmisă în lista de parametri a subprogramelor. Totuși, *n* a fost transmis ca dată de intrare subprogramului *pozMin* deoarece am folosit forma generală a subprogramului de determinare a poziției minimului, în care trebuie să apară numărul de elemente pentru care se calculează minimul și poziția acestuia;
- variabila *suma* din subprogramul *diag* este necesară numai pentru prelucrările din acest subprogram. Această variabilă a fost declarată în subprogram și este variabilă locală pentru acesta;
- dacă valoarea sumei minime nu trebuia afișată, atunci variabila *Val* putea fi declarată în interiorul subprogramului *pozMin* (deci ar fi fost o variabilă locală pentru el, la fel ca variabila *i*). Fiind declarată în programul principal (apelant) ca variabilă globală, valoarea ei s-a păstrat și a putut fi afișată;
- variabilele care apar în lista parametrilor actuali trebuie declarate în programul apelant; dar variabilele care apar în lista parametrilor formali nu trebuie declarate. Totuși, în programul

sumaMinimă variabila *V*, parametru formal pentru subprogramul *pozMinim* a fost declarată în programul apelant (principal). Motivul: ea este folosită și ca parametru actual pentru apelarea subprogramului *pozMinnim*.

TEME

1. Variabile locale și variabile globale

Fie programul de mai jos:

```
început program test
  variabile a, b, c
  început subprogram  S1 (adresa y)
    .....
    returnează y
    .....
  sfârșit S1
  început subprogram S2 (x, adresa t)
    variabile a, y
    început subprogram  S3 (adresa y)
      variabile b
      .....
      returnează y
      .....
    sfârșit S3
      .....
    returnează t
      .....
  sfârșit S2
    .....
sfârșit test
```

Presupunem că plasăm următoarele instrucțiuni în blocul de instrucțiuni executabile al subprogramului menționat. Precizați care dintre instrucțiuni va fi corect/incorect plasată și de ce:

a) în subprogramul S3, instrucțiunea a ← b
b) în programul test, instrucțiunea **returnează** S3(3.14)
c) în programul test, instrucțiunea a ← S2(10)
d) în subprogramul S1, **apelează** S3(d)
e) în subprogramul S1, c ← S2(c)
f) în programul test, instrucțiunea **apelează** S1(z)
g) în programul test, instrucțiunea **apelează** S1(S2(3))
h) în programul test, instrucțiunea **apelează** S2(c)
i) în subprogramul S2, **apelează** S1(y)
j) în subprogramul S1, y ← S2(c)

2. Variabile locale și variabile g

Care dintre afirmațiile de mai
a) nu este permisă utilizarea
b) nu este recomandabilă util
c) variabilele globale pot fi f
d) variabilele globale pot fi f
e) datele de intrare ale subpr

3. Reguli de lucru cu subprogr

Care dintre afirmațiile de mai
a) fie S1 un subprogram din
S1; atunci S2 nu poate conți
b) constantele utilizate într-
meniul lor de vizibilitate ca și
c) într-un subprogram putem
d) într-un program cu subpr
me, numai dacă variabila este

4. Corectarea erorilor în subpr

Fie programul de mai jos:

```
început program test2
  variabile a, b, c
  început subprogram S1(x,
    variabile a, y
    început subprogram S2(
      variabile b
      .....
      returnează y
      .....
    sfârșit S3
    .....
    returnează w
    .....
  sfârșit S1
  început subprogram S3 (a
    .....
    returnează y
    .....
  sfârșit S3
    .....
sfârșit test
```

2. Variabile locale și variabile globale

Care dintre afirmațiile de mai jos este corectă? Corectați afirmațiile greșite:

a) nu este permisă utilizarea variabilelor globale în interiorul unui subprogram;
b) nu este recomandabilă utilizarea variabilelor globale în interiorul unui subprogram;
c) variabilele globale pot fi folosite exclusiv pentru citirea datelor de intrare în program;
d) variabilele globale pot fi folosite exclusiv pentru citirea datelor de intrare în subprogram;
e) datele de intrare ale subprogramelor nu pot fi transmise decât prin variabile globale.

3. Reguli de lucru cu subprogramele

Care dintre afirmațiile de mai jos este corectă/greșită; justificați răspunsul:

a) fie S1 un subprogram din programul principal P și S2 un subprogram definit în interiorul lui S1; atunci S2 nu poate conține la rândul său alt subprogram;
b) constantele utilizate într-un program cu subprograme urmează aceleași reguli privind domeniul lor de vizibilitate ca și variabilele;
c) într-un subprogram putem declara și o variabilă și o constantă sub același nume;
d) într-un program cu subprograme putem declara și o variabilă și o constantă sub același nume, numai dacă variabila este globală iar constanta este locală.

4. Corectarea erorilor în subprograme

Fie programul de mai jos:

```
început program test2
 variabile a, b, c
 început subprogram S1(x, adresa w)
  variabile a, y
  început subprogram S2(adresa y)
   variabile b
   .....
   returnează y
   .....
  sfârșit S3
  .....
  returnează w
  .....
 sfârșit S1
 început subprogram S3 (adresa y)
   .....
  returnează y
   .....
 sfârșit S3
   .....
sfârșit test
```

Presupunem că plasăm următoarele instrucțiuni în blocul de instrucțiuni executabile al sub-programului menționat. Precizați care dintre instrucțiuni va fi corect/incorect plasată și de ce (atenție la domeniul de vizibilitate al variabilelor):

a) în subprogramul S2, instrucțiunea a ← b
b) în programul test, instrucțiunea **apelează** S2(3.14)
c) în programul test, instrucțiunea a ← S1(10)
d) în subprogramul S3, **apelează** S2(3.14)
e) în subprogramul S3, c ← S1(c)
f) în programul test, instrucțiunea **apelează** S3(a)
g) în programul test, instrucțiunea **apelează** S1(S3(z))
h) în programul test, instrucțiunea **apelează** S1(c)
i) în subprogramul S1, instrucțiunea **apelează** S3(y)
j) în subprogramul S2, instrucțiunea y ← S1(c)

APLICAȚIE

TAPETAREA APARTAMENTULUI

Să se calculeze suprafața de tapet necesară pentru renovarea unei sufragerii, a unui dormitor și a unei bucătării (toate de formă paralelipipedică).

1. Analiza problemei

- Date de intrare:
  - lungimile pereților sufrageriei (**S1, S2**), dormitorului (**D1, D2**) și ai bucătăriei (**B1, B2**),
  - înălțimea pereților (**H**),
  - lățimea sulului de tapet (**T**).
- Date de ieșire:
  - numărul de metri de tapet necesari pentru sufragerie (**NS**), dormitor (**ND**), bucătărie (**NB**),
  - numărul total (**NR**).
- Condiții și relații importante

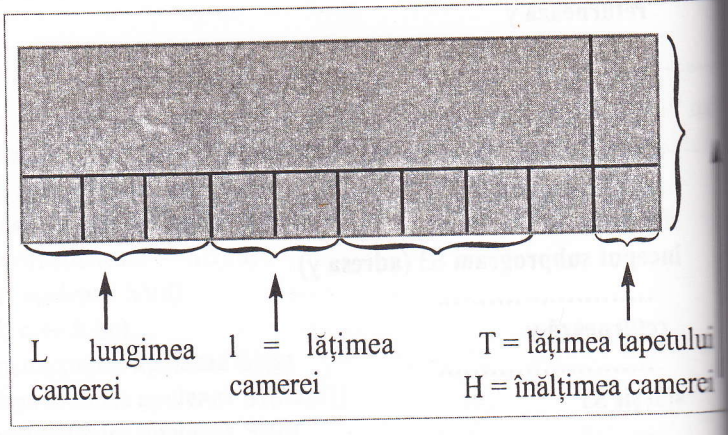


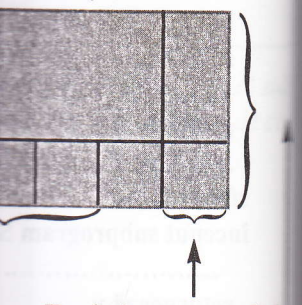
Figura 41. Acoperirea pereților camerei cu tapet

nstrucțiuni executabile al sub-
rect/incorect plasată și de ce

unei sufragerii, a unui dormi-

2) și ai bucătăriei (**B1, B2**),

rmitor (**ND**), bucătărie (**NB**),



T = lățimea tapetului
H = înălțimea camerei

or camerei cu tapet

Începem cu sufrageria: este suficient să calculăm P = 2*(S1+S2) = perimetrul camerei și să-l împărțim la T = lățimea sulului de tapet pentru a afla m = P / T = câte bucăți de tapet de lungime egală cu înălțimea camerei sunt necesare. Apoi, NS = m * H = numărul de metri de tapet necesari pentru sufragerie. Analog obținem numărul de metri de tapet necesari pentru dormitor și pentru bucătărie. În final, calculăm numărul total NR = NS + ND + NB.

2. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: S1, S2, D1, D2, B1, B2, H, T.
Pasul 2. Se calculează P ← 2 * (S1 + S2); m ← P / T; NS ← m * H.
Pasul 3. Se calculează P ← 2 * (D1 + D2); m ← P / T; ND ← m * H.
Pasul 4. Se calculează P ← 2 * (B1 + B2); m ← P / T; NB ← m * H.
Pasul 5. Se calculează NR ← NS + ND + NB.
Pasul 6. Se afișează NS, ND, NB, NR.

3. Reprezentarea algoritmului

Metoda I: Un program fără module

```
început tapet1
variabile S1,S2,D1,D2,B1,B2,H,T,NS,ND,NB,NR,P,m
citește S1,S2,D1,D2,B1,B2,H,T
P ← 2*(S1 + S2)
m ← P/T
NS ← m*H
P ← 2*(D1 +D2)
m ← P/T
ND ← m*H
P ← 2*(B1 + B2)
m ← P/T
NB ← m*H
NR ← NS+ND+NB
scrie NS,ND,NB,NR
sfârșit tapet1
```

Metoda II: Un program modularizat

```
început tapet2
variabile S1,S2,D1,D2,B1,B2,H,T,NS,ND,NB,NR
început subprogram calcul(L;l;adresa:N)
variabile P,m
P ← 2*(L+l)
m ← P/T
N ← m*H
returnează N
sfârșit calcul
citește S1,S2,D1,D2,B1,B2,H,T
apelează calcul(S1;S2;NS)
```

apelează calcul(D1;D2;ND)
apelează calcul(B1;B2;NB)
NR ← NS+ND+NB
scrie NR,NS,ND,NB
sfârșit tapet2

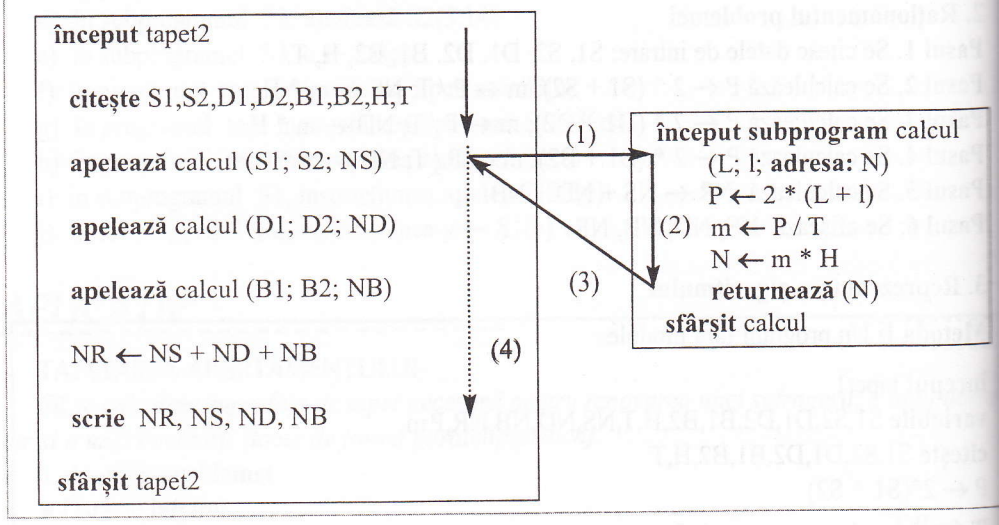


Figura 42. Programul tapet2 – fluxul de execuție

În figura 42 este evidențiat fluxul de execuție a programului:

(1) controlul este transferat subprogramului;

(2) se execută instrucțiunile subprogramului în același mod în care s-ar executa dacă ar fi vorba de un program propriu-zis;

(3) după parcurgerea întregului flux de execuție al subprogramului, controlul (și rezultatele) se transferă programului apelant; execuția acestuia continuă cu instrucțiunea aflată imediat după instrucțiunea care apelează subprogramul;

(4) fiecare apel determină reluarea fluxului (1), (2), (3).

TEME

1. Fluxul de execuție

Reprezentați stiva sistem corespunzător fluxului de execuție program – subprogram din figura 42.

2. Rezolvarea problemelor cu ajutorul subprogramelor

Scrieți câte un program cu subprograme pentru rezolvarea fiecăreia dintre următoarele cerințe:

a) Să se determine numărul *p* de numere prime dintr-un șir de *n* numere naturale introduse de la tastatură. Să se verifice dacă *p* este număr prim și să se afișeze un mesaj corespunzător

Exemplu: pentru n = 5, șirul de numere este: 25, 3, 42, 29, 7. Se determină p = 3 și se afișează mesajul: *3 este număr prim.*

b) Să se calculeze numerele „mistice" mai mici decât un număr $n \in N$ dat ($x \in N$ se numește număr „mistic" dacă suma cifrelor care îl compun este egală cu produsul lor; exemplu: 22, 13131).

c) Să se calculeze divizorii primi ai unui număr $n \in N$ dat.

d) Să se calculeze numerele perfecte mai mici decât un număr $n \in N$ dat ($x \in N$ se numește număr perfect dacă suma divizorilor săi primi este egală cu produsul lor; exemplu: 6 este un număr perfect deoarece 1+2+3=2*3).

e) Fie M o matrice cu *n* linii și *m* coloane. Să se afișeze în ordine crescătoare elementele de pe liniile pare.

f) Fie M o matrice cu *n* linii și *m* coloane. Pentru fiecare linie, să se afișeze cel mai mare divizor comun dintre *max* și *min*, unde *max* reprezintă cea mai mare valoare de pe linie iar *min* reprezintă cea mai mică valoare de pe linie.

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Etapa: *Proiectare* (stabilirea unităților de program)

Cerințe:

1. Organizați unitățile de program necesare rezolvării modulare a prelucrărilor identificate în etapa de analiză.

2. Completați documentația proiectului cu descrierea modulelor și reprezentarea relațiilor dintre acestea.

4. IMPLEMENTAREA SUBPROGRAMELOR ÎN LIMBAJELE DE PROGRAMARE

A. SUBPROGRAME ÎN LIMBAJUL PASCAL: FUNCȚII ȘI PROCEDURI

În limbajul ***Pascal*** se utilizează două categorii de subprograme:

- funcții,
- proceduri.

Vom prezenta asemănările și deosebirile dintre aceste două categorii de subprograme cu ajutorul programului de calculare a ariei unui dreptunghi

Definiție

Funcție =

= un subprogram care primește oricâte valori ca date de intrare dar returnează ca rezultat o singură și numai una.

Declarare:

function nume(par_formal$_1$,…,par_formal$_n$:tip$_1$;…; par_formal$_1$,…,par_formal$_m$:tip$_k$): tip;

Apelare:

x:= nume(par_actual$_1$;par_actual$_2$;…;par_actual$_w$);

sau:

write(nume(par_actual$_1$;par_actual$_2$;…;par_actual$_w$));

🕮 Definiție

Procedură =

= un subprogram care primește oricâte valori ca date de intrare și poate returna ca rezultat o valoare, mai multe valori sau niciuna (poate efectua o prelucrare care să nu aibă ca efect obținerea unei valori).

Declarare:

procedure nume([*var*] par_formal$_1$,…,par_formal$_n$:tip$_1$;…; [*var*] par_formal$_1$,…, par_formal$_m$:tip$_k$);

Apelare:

nume(par_actual$_1$;par_actual$_2$;…;par_actual$_w$);

🔔 Atenție

Spre deosebire de pseudocod, în limbajul ***Pascal:***

- cuvântul rezervat pentru transmiterea parametrilor prin adresă este *var*;
- apelarea subprogramelor (ca și returnarea rezultatelor) nu necesită utilizarea niciunui cuvânt rezervat (altfel spus, cuvintele rezervate **apelează** și **returnează** folosite în pseudocod nu au un corespondent în limbajul ***Pascal***).

	Funcții		***Proceduri***
Declarare	**function** A (L:**Real**; l:**Real**): **Real**;		**procedure** arie(L:**Real**; l:**Real**; **var** A: **Real**); eventual
Apelare	v:=A (Lung;H);	write (A(Lung;H));	arie(Lung;H; suprafata);

Observații

- Pentru identificarea fiecărui tip de subprogram se utilizează cuvinte rezervate diferite: **function** și respectiv **procedure**.
- Declarația *var* este indusă între paranteze drepte deoarece este opțională.

• Lista de parametri formali a procedurii trebuie să includă un număr de parametri formali transmiși prin adresă egal cu numărul de rezultate returnate de procedură către programul apelant. Lista de parametri formali a funcției include parametrii transmiși prin valoare (datele de intrare) sau prin referință (dacă există date de intrare-ieșire); valoarea determinată – rezultatul – va fi returnată prin *numele* funcției. De aceea:

– „funcția are tip" (și anume un tip simplu, nestructurat);

– în blocul de instrucțiuni executabile al funcției trebuie să existe cel puțin o instrucțiune de atribuire având ca membru stâng **numele** funcției.

• Funcția se apelează prin nume și listă de parametri actuali în interiorul unei expresii.

• Procedura se apelează prin nume și listă de parametri actuali într-o instrucțiune de sine stătoare.

Exemplu: Funcții și proceduri în limbajul ***Pascal***

Fiind dat numărul real a și numărul întreg n, să se calculeze a^n.

1. Analiza problemei

• Date de intrare:

baza (a) și exponentul (n).

Date de ieșire:

valoarea a^n.

2. Modularizarea rezolvării

Vom utiliza procedura *putere1* și funcția *putere2* (pentru a ilustra transformarea unei proceduri într-o funcție și reciproc).

Vom valida datele de intrare, tratând cazul 0^0 într-un mod particular: $0^0=0$ (cazul bazei egale cu 1 nu va fi tratat separat!).

3. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: a, n.
Pasul 2. Dacă a = 0 atunci r ← 0 și salt la **Pasul 9**.
Pasul 3. Dacă n = 0 atunci r ← 1 și salt la **Pasul 9**
Pasul 4. nn ← n; dacă nn < 0 atunci nn ← – nn
Pasul 5. i ← 1.
Pasul 6. r ← r * a și i ← i + 1
Pasul 7. Dacă i ≤ nn atunci se reia **Pasul 6**.
Pasul 8. Dacă n < 0 atunci r ← 1/ r
Pasul 9. Se scrie r.

4. Implementarea algoritmului în limbajul *Pascal*

```
program ridicareLaPutere;
var a,r:real;
    n:integer;
procedure putere1(baza:real; expo:integer; var rez:real);
{ rez – parametru transmis prin adresă }
var e,i:integer;
```

```
  begin
   if (baza = 0.0) then rez:=0.0
    else
     if (expo=0) then rez:=1.0
      else
       begin
       rez:= baza;
       if expo<0 then e:=-expo else e:=expo;
       for i:=2 to e do rez:=rez*baza;
       if expo<0 then rez:=1/rez;
       end;
  end; {procedura putere1}
 function putere2(baza:real; expo:integer):real; {funcție de tip real}
  var e,i:integer;
     rez:real;
  begin
   if (baza = 0.0) then putere2:=0.0
    else
     if (expo=0) then putere2:=1.0
      else
       begin
       rez:= baza;
       if expo<0 then e:=-expo else e:=expo;
       for i:=2 to e do rez:=rez*baza;
       if expo<0 then rez:=1/rez;
       putere2:= rez;
       end;
  end; {functia putere2}
  begin {program principal}
   write('a ='); readln(a);
   write('n ='); readln(n);
   putere1(a,n,r);
   write('Calculul puterii cu procedura putere1: ');
   writeln(a:4:2,' la puterea ',n:2,' este ',r:8:3);
   write('Calculul puterii cu  functia  putere2: ');
   writeln(a:4:2,'la puterea ',n:2,' este',putere2(a,n):8:3)
  end.
```

5. Analiza programului

Identificatorul *putere2* **nu** desemnează o variabilă (având același nume și tip ca și funcția), ci este numele funcției și **acestuia** i se atribuie valoarea variabilei **rez.** Acesta este motivul pentru care, în blocul de instrucțiuni executabile al funcției **care nu se apelează pe ea însăși**, numele funcției nu poate apărea în membrul drept al unei atribuiri: este necesară utilizarea unei

variabile locale, auxiliare. Tabelul de mai jos prezintă funcțiile *putere2* și *putere3* ca exemplu și contraexemplu de subprograme de tip funcție.

Exemplu de funcție Pascal: putere2	**Contraexemplu de funcție Pascal: putere3**

function putere2(baza:**real**;
expo:**integer**):**real**;
var e,i:**integer**;
rez:**real**;
begin
if (baza=0.0) **then** putere2:=0.0
else
if (expo=0) **then** putere2:=1.0
else
begin
rez:= baza;
if expo<0 **then** e:=-expo **else** e:=expo;
for i:=2 **to** e **do** rez:=rez*baza;
if expo<0 **then** rez:=1/rez;
putere2:= rez;
end;
end; {functia putere2}
	function putere3(baza:**real**,
	expo:**integer**):**real**;
	var e,i:**integer**;
	begin
	if (baza=0.0) **then** putere3:=0.0
	else
	if (expo=0) **then** putere3:=1.0
	else
	begin
	putere3:=baza;
	if expo<0 **then** e:=-expo **else** e:= expo;
	for i:=2 to e do putere3:= putere3*baza
	if expo<0 then putere3:=1/putere3;
	end
	end; {functia putere3}

TEMĂ

De la procedură la funcție

Transformați următoarea procedură într-o funcție:

```
procedure Verificare (err,toleranta:real; var accept:boolean);
begin
if err<toleranta then
  accept:=true
  else
  accept:=false;
end;
```

Observație

Tipul unui parametru formal nu poate fi un tip anonim:

Exemplu:	***type*** matrice=***array***[1..10,1..20] ***of real***; ***function*** maxim(n,m:***integer***; A:matrice):***real***;
Contraexemplu	***function*** maxim(n,m:***integer***; A:***array***[1..10,1..20] ***of real***):***real***;

TEME

Proceduri sau funcții?

1. Fie două numere raționale r1 și r2; se cere să se afișeze: suma, produsul și câtul celor două numere. *Scrieți și analizați cele două soluții posibile: un program cu subprograme de tip funcție, respectiv cu subprograme de tip procedură (pentru fiecare operație).*

2. Fie **M** o matrice pătratică de ordin **n** > 1 cu elemente numere naturale; se cere să se afișeze numerele prime (și poziția lor în matrice) aflate ca valoare între cel mai mic număr de pe diagonala principală și cel mai mare număr de pe diagonala secundară. *Care dintre module poate fi transformat numai într-o funcție/numai într-o procedură/fie într-o procedură/fie într-o funcție. Argumentați.*

3. Fie vectorul **V** cu **n** > 1 componente, numere naturale; *se cere să se afișeze:*

a) numere pare care se găsesc pe poziții de index impar în **V**;
b) numerele prime din **V**;
c) cifrele numărului de pe ultima poziție din vector, în ordine inversă.

Domeniul de vizibilitate al variabilelor

4. *Examinați programul **Pascal** de mai jos din punct de vedere al domeniului de vizibilitate al variabilelor. Completați (după modelul oferit pe prima linie) toate liniile din tabelul furnizat, indicând tipul de date al variabilelor (prin inițiala tipului: I pentru Integer, R pentru real etc.) și valorile constantelor, în conformitate cu cerințele fiecărui bloc de instrucțiuni executabile. Dacă un identificator nu este definit într-un anumit bloc, atunci introduceți în celula respectivă codul N/A.*

```
program domeniu;
 const i=10;
 var a,b:integer;
    c,f:real;
    d,e:char;
      g:boolean;
 procedure P1(a,e:real; c:boolean);
  const k=2.1;
  var  f:boolean;
  begin {P1}
  .....
  end; {P1}
 procedure P2(c:integer; g:real);
  const b=10;
  var  a:boolean;
       e:char;
  procedure P3(a:char; b:real);
  const d=3.14;
  var i,f:integer;
  begin {P3}
  .....
  end; {P3}
 procedure P4(a:real);
  const e='a';
  var g,k:integer;
  begin {P4}
  .....
  end; {P4}
 begin {P2}
  .....
 end; {P2}
procedure P5(e,g: intege
 const k=40;
 var  i:real;
 begin {P5}
  .....
 end; {P5}
begin {program principal
  .....
end.
```

Domeniul	*a*	*b*
Program princ.	I	I
P1		
P2		
P3		
P4		
P5		

B. SUBPROGRAME ÎN L

Limbajul *C/C++* admite
gramul principal este o fun
Declarare:

```
tip nume(tip1 [&
```

Declarația & este inclusă în
Apelare

```
v=nume(par_actual
```

```
  const d=3.14;
  var i,f:integer;
  begin {P3}
  ..................
  end; {P3}
 procedure P4(a:real);
  const e='a';
  var g,k:integer;
  begin {P4}
  ..................
  end; {P4}
 begin {P2}
  ..................
 end; {P2}
procedure P5(e,g: integer; d:boolean);
 const k=40;
 var  i:real;
 begin {P5}
  ..................
 end; {P5}
begin {program principal}
  ..................
end.
```

Domeniul	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*i*	*k*
Program princ.	I	I	R	C	C	R	B	10	N/A
P1									
P2									
P3									
P4									
P5									

B. SUBPROGRAME ÎN LIMBAJUL C/C++

Limbajul *C/C++* admite un singur tip de subprograme: subprograme de tip funcție (însuși programul principal este o funcție cu un nume rezervat: **main**).

Declarare:

tip nume(tip1 [&] par_formal$_1$,..., tip$_m$ [&] par_formal$_m$);

Declarația & este inclusă între paranteze drepte deoarece este opțională.

Apelare

v=nume(par_actual$_1$,..., par_actual$_m$); sau

nume(par_actual$_1$,..., par_actual$_m$);

În limbajul C/C++ există două categorii de funcții:

• funcții „cu tip" – acestea transmit către programul apelant o valoare atribuită numelui funcției; dacă tipul expresiei din instrucțiunea

return expresie;

nu este identic cu cel al funcției, atunci se face automat conversia. O astfel de funcție se apelează într-o expresie;

• funcții „fără tip" – acestea nu transmit către programul apelant nicio valoare; în acest caz, tipul funcției este **void**, rezultatele prelucrărilor sunt depuse în variabile globale (și astfel programul apelant are acces la ele) iar revenirea în programul apelant se face fie la întâlnirea instrucțiunii

return;

fie după executarea tuturor instrucțiunilor din corpul funcției. O astfel de funcție se apelează printr-o instrucțiune de sine stătătoare.

În descrierea și apelul funcțiilor trebuie respectate următoarele reguli:

• numele funcției este precedat de tipul funcției;

• numele fiecărui parametru formal este precedat de tipul parametrului;

• transmiterea parametrilor prin adresă este precizată prin operatorul **&** care precede numele parametrului formal, de exemplu:

double arie(**double** L, **double** l, **double &** A));

• în blocul de instrucțiuni al funcției există o instrucțiune specială pentru returnarea rezultatului: **return** expresie; sau **return**;

return A;

Exemplul 1: Verificarea parității sumei a două numere întregi.

1. Analiza problemei

• Date de intrare:

numerele întregi **a, b.**

• Date de ieșire:

un mesaj care indică dacă suma celor două numere este pară sau nu.

2. Modularizarea rezolvării

Varianta I

Vom folosi o funcție care returnează o valoare (restul împărțirii sumei la 2).

Varianta II

Vom folosi o funcție care nu returnează nicio valoare (variabila în care se calculează restul este declarată variabilă globală):

3. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: a, b.

Pasul 2. Se apelează subprogramul de determinare a parității sumei celor două numere.

Pasul 3. Dacă restul returnat de subprogram este zero atunci se afișează mesajul "Suma este pară", altfel se afișează mesajul "Suma este impară".

I. Program cu funcție returnează o valoa

```
#include<stdlib.h>
#include<conio.h>

int par(int a, int b)
{int S;
S=(a+b)%2;
return S;}
void main (void)
{int a,b,S;
cout<<"Dați întregii a și b";
cin>>a>>b;
S=par(a,b);
if (S == 0) cout<<"Suma celo
numere este para";
  else cout<<"Suma celor do
este impara";}
```

Exemplul 2: Interschimba

În foarte multe prelucrări, e
memorie, fie acestea *a* și *b*. D
valorile celor două locații de m
mite prin interschimbare. Subp
două locații de memorie.

Fu

```
#include<stdlib.h>
#include<conio.h>

void interschimb (int
 {int aux;
  aux=x; x=y; y=aux;}
void main (void)
 {int a,b;
 cout<< "se introduc
// are loc interschimb
   interschimb (a, b);
   cout << "se afișeaz
   cout << a << " " <<
   }
```

4. Implementarea algoritmului

I. Program cu funcție care returnează o valoare	II. Program cu o funcție care nu returnează nicio valoare

#include<stdlib.h>
#include<conio.h>
 	
int par(int a, int b)
{int S;
S=(a+b)%2;
return S;}
void main (void)
{int a,b,S;
cout<<"Dați întregii a și b";
cin>>a>>b;
S=par(a,b);
if (S == 0) cout<<"Suma celor doua numere este para";
else cout<<"Suma celor doua numere este impara";}
	#include<stdlib.h>
	#include<conio.h>
	int S;
	void par (int a, int b)
	
	{S=(a+b)%2;}
	
	void main (void)
	{int a,b;
	cout<<"Se introduc a și b";
	cin>>a>>b;
	par(a,b);
	if (S==0) cout<<"Suma celor doua numere este para";
	else cout<<"Suma celor doua numere este impara";}

Exemplul 2: Interschimbarea a două valori *a* și *b*.

În foarte multe prelucrări, este necesară interschimbarea valorilor reținute în două locații de memorie, fie acestea *a* și *b*. De aceea, este util să realizăm un subprogram care să primească valorile celor două locații de memorie și să transmită programului principal noile valori rezultate prin interschimbare. Subprogramul necesită doi parametri care reprezintă adresele celor două locații de memorie.

Funcție cu parametri de tip adresă

#include<stdlib.h>
#include<conio.h>
 	
void interschimb (int &x, int &y)
{int aux;
aux=x; x=y; y=aux;}
void main (void)
{int a,b;
cout<< "se introduc valorile";
// are loc interschimbarea valorilor
interschimb (a, b);
cout << "se afișează noile valori";
cout << a << " " << b;
}

TEME

Utilizarea subprogramelor în limbajul C/C++

1. Fie două numere raționale r1 și r2; se cere să se afișeze: suma, produsul și câtul celor două numere. Scrieți și analizați un program cu subprograme (pentru fiecare operație câte un subprogram).
2. Fie M o matrice pătratică de ordin n > 1 cu elemente numere naturale; se cere să se afișeze numerele prime (și poziția lor în matrice) aflate în intervalul dintre cel mai mic număr de pe diagonala principală și cel mai mare număr de pe diagonala secundară. Realizați un program cu subprograme. Descrieți și argumentați tipul fiecărui subprogram.
3. Examinați programul C/C++ de mai jos din punct de vedere al domeniului de vizibilitate al variabilelor.

```
#include<iostream.h>
char a,b; int c; float d;
void P1 (float y)
{float b;
————————}
void P2(char y)
{————————}
char F1(int x)
{float a; char y;
P1(y);
return y;}
void main (void)
{————————}
```

Presupunem că plasăm următoarele instrucțiuni în blocul de instrucțiuni executabile al subprogramului menționat. *Indicați care dintre instrucțiuni va fi corect/incorect plasată și de ce:*

a) în program, P2(F1(c));
b) în subprogramul F1, P1(a);
c) în subprogramul P2, P1(d);
d) în program, a=F1(c);
e) în subprogramul P1, P2(b).

C. EXERCIȚII CU SUBPROGRAME

1. Codificați în limbajul de programare studiat următoarele secvențe și completați tabelul de variație pentru a determina valoarea variabilelor pe parcursul execuției.

```
început program
 variabile a,b,c,d,
 început subprog
  variabile a
  a←1; b←7; x←
  returnează z,u,v
 sfârșit P1;
 a←4; b←5; c←1
 apelează P1(a;b;
 scrie a,b,c,d,e
 apelează P1(1;2;
 scrie a,b,c,d,e
sfârșit calcul
```

```
început program
 variabile x,y
 început subprog
  variabile i,s
  s ← 0;
  pentru i = 1la y
   s ← s+i+x
   x ← -1
  sfârșit pentru
  F1 ← s
 sfârșit F1
 x ← 0; y ← 1
 cât timp y <= 2
  scrie x,y
  x ← x–F1(y);
  y ← y+1;
 sfârșit cât timp
sfârșit opuse
```

```
început program ve
  început subprog
  dacă v1>v2
      atunci
        scrie v1,v2
        v3 ← v3+1
      altfel
        scrie v2,v1
```

```
început program calcul
 variabile a,b,c,d,e
 început subprogram P1(x;y; adresa:z;u;v)
  variabile a
  a←1; b←7; x←y; z←a+x; v←z+b
  returnează z,u,v
 sfârșit P1;
 a←4; b←5; c←12; d←b; e←c
 apelează P1(a;b;c;d;e)
 scrie a,b,c,d,e
 apelează P1(1;2;a;b;c)
 scrie a,b,c,d,e
sfârșit calcul
```

a	b	c	d	e

```
început program opuse
 variabile x,y
 început subprogram F1(x)
  variabile i,s
  s ← 0;
  pentru i = 1la y execută
   s ← s+i+x
   x ← -1
  sfârșit pentru
  F1 ← s
 sfârșit F1
 x ← 0; y ← 1
 cât timp y <= 2
  scrie x,y
  x ← x–F1(y);
  y ← y+1;
 sfârșit cât timp
sfârșit opuse
```

x	y	s	i

```
început program verificare
 început subprogram verificăValori(date de intrare: v1; v2;v3; date de ieșire:v3)
 dacă v1>v2
    atunci
      scrie v1,v2
      v3 ← v3+1
    altfel
      scrie v2,v1
```

```
        v3 ← v3-1
    sfârșit dacă
  sfârșit verificăValori
  contor←0
  citește index
  pentru i=1 la index execută
   citește v1,v2,v3,v4
   contor←contor+1
   citește v5
   cât timp v3<v5
      v3←v3+2
   sfârșit cât timp
   v5←v5+v3
   dacă v3 este par
       atunci
        apelează verificăValori(v2;v4;v5)
       altfel
        apelează verificăValori(v1;v3;v5)
   sfârșit dacă
  sfârșit pentru
  scrie v1,v2,v3,v4,v5
 sfârșit program
```

v1	v2	v3	v4	v5

2. Corectarea erorilor

– Eliminați erorile din programele de mai jos:

a)

VARIANTA *PASCAL*	VARIANTA *C/C++*
program erori; **const** limita=10; **var** index:**integer**; **function** Func(cheie:**char**): **boolean**; **var** sem:**boolean**; cod:**char**; **begin** {Func}	**#include<iostream.h>** **#include<conio.h>** **int** limita=10; **int** Func(**char** cheie) { **int** sem,cod; **cout**<<"Introduceți codul ";

```
  sem:=false;
  write('Introduceți codul');
  read (cod); writeln;
  if cod=cheie then        sem
  Func:=sem
 end {Func}
begin {program principal}
 for index:=1 to limita
 if (func(chr(index)) then
   writeln ('Da');
 end; {if}
end. {program principal}
```

VARIANTA P

```
program erori2;
var a,b,i:integer;
procedure P1(var x:integ
 begin {P1}
  x=y
 end {P1}
begin {program principal}
set s to false
while not s
 begin
  readln(a and b);
  if (a>b)
   then S1 a, b);
   else S1(b,a);
  for i=1 to b do
   begin
    a:=a-(b+i);
    writeln('a+b=,a+b)
   end for;
end. {program principal}
```

III. Stive și cozi

a) Realizați un program

b) Realizați un program

v2	v3	v4	v5

RIANTA C/C++

```
a.h>
>
e)
ți codul ";
```

```
sem:=false;
write('Introduceți codul");
read (cod); writeln;
if cod=cheie then        semaf:=true
Func:=sem
end {Func}
begin {program principal}
for index:=1 to limita
if (func(chr(index)) then
  writeln ('Da');
end; {if}
end. {program principal}
```

```
cin>>cod; cout<<endl;
if(cod==cheie)
  semaf=1;
return sem;
}
void main (void)
for (int index=1; index<=limita;index++)
{if (func(char(index)))
cout<<"Da";
}
getch();
}
```

VARIANTA PASCAL	VARIANTA C/C++

program erori2;
var a,b,i:integer;
procedure P1(var x:integer, y:integer);
begin {P1}
x=y
end {P1}
begin {program principal}
set s to false
while not s
begin
readln(a and b);
if (a>b)
then S1 a, b);
else S1(b,a);
for i=1 to b do
begin
a:=a-(b+i);
writeln('a+b=,a+b)
end for;
end. {program principal}
	#include<iostream.h>
	
	int a,b,i;
	void P1(int x, int y)
	{x=y}
	void main (void)
	{s=0;
	while (!s)
	{cin>>a>>b;
	if(a>b) S1(a,b);
	else S1(b, a)
	for (i=1;i<=b,i++)
	{a=a-(b+i);
	cout<<'a+b= '<<a+b;}}

III. Stive și cozi

a) Realizați un program cu subprograme pentru simularea unei stive.

b) Realizați un program cu subprograme pentru simularea unui fir de așteptare.

Utilizarea funcțiilor predefinite

4. Scrieți câte un program care să utilizeze funcțiile și procedurile standard enumerate în Anexa 2/Anexa 4.
5. Identificați funcțiile predefinite ce se pot folosi în programul de calculare a energiei cinetice și în problema traiectoriilor (pag. 65 și pag. 70).
6. Se consideră un automobil într-o curbă de rază r; dându-se unghiurile pantelor α și φ, se cere să se scrie un program care să afișeze viteza maximă și viteza minimă cu care poate ataca automobilul curba în condiții de siguranță. Se cunosc următoarele formule:

$$v_M = \sqrt{r \cdot g \cdot tg(\alpha + \varphi)} \quad , \quad v_m = \sqrt{r \cdot g \cdot tg(\alpha - \varphi)}$$

Ce subprograme predefinite se pot folosi?

5. ȘIRURI DE CARACTERE

A. PARTICULARITĂȚI DE MEMORARE A ȘIRURILOR DE CARACTERE

Cum procedăm pentru a citi și afișa, pe rând, numele inventatorilor primelor calculatoare mecanice, electromecanice, electronice: John Napier, Wilhelm Schickard, Blaise Pascal, Gottfried Wilhelm Leibnitz, Charles Babbage, Konrad Zuse, S.P. Eckert, J.W. Mauchly ș.a.

Să începem cu C. Babbage: o primă soluție constă în utilizarea unui vector cu cel puțin 10 componente de tipul predefinit **char**. Putem citi, prelucra și afișa tot vectorul sau numai anumite elemente ale sale (a se vedea figura 43).

indice în vectorul s	0	1	2	3	4	5	6	7	8	9
caracter tastat		**C**	**.**	**B**	**a**	**b**	**b**	**a**	**g**	**e**
valoare în memorie	9	067	046	066	097	098	098	097	103	101

(a) varianta ***Pascal***

indice în vectorul **s**	0	1	2	3	4	5	6	7	8	9
caracter tastat	**C**	**.**	**B**	**a**	**b**	**b**	**a**	**g**	**e**	**null**
valoare în memorie	067	046	066	097	098	098	097	103	101	000

(a) varianta ***C/C++***

Figura 43. Memorarea șirurilor de caractere

Se observă că în locațiile din memoria internă asociate componentelor vectorului sunt depuse codurile *ASCII* (American Standard Code for Information Interchange (ASCII) este codul universal acceptat pentru memorarea internă a caracterelor) ale caracterelor din șir: s[1] conține valoarea 067 = codul *ASCII* al caracterului 'C', s[3] conține valoarea 066 = codul *ASCII* al caracterului 'B', s[5] și s[6] conțin valoarea 098 = codul *ASCII* al caracterului 'b' etc. (Anexa 7)

ÎN PASCAL	ÎN C/C++
în primul octet (atenție: s[0] și nu s[1] este prima componentă a vectorului!) este depusă automat lungimea efectivă a șirului (aici: 9).	ultimul octet (s[9], pentru că indexarea se face de la 0!) conține, conform convenției, valoarea 0 = codul *ASCII* al caracterului null

LIMBAJUL PASCAL	LIMBAJUL C/C++
program sir; **var** s:**array**[1..50] of **char**; n,i: **integer**; **begin** **writeln**('n='); **readln**(n); **for** i:=1 **to** n **do read** (s[i]); **writeln**('Sirul citit este:'); **write** (s); **end.**	**#include<iostream.h>** **void main (void)** {**char** s[50]; **int** n, i; **cout**<<"n="; **cin**>>n; **cout**<<"Introduceti sirul."<<**endl**; **for** (i=0; i<n; i++) **cin**>>s[i]; s[n]=0; **cout**<<"Sirul este:"<<**endl**; **for** (i=0;i<n;i++)**cout**<<s[i];}

Atenție

În limbajul *Pascal* am putut afișa vectorul de caractere ca pe o variabilă independentă (nu am mai recurs la indexare, așa cum am făcut la citire).

În limbajul *C/C++*, dacă vectorul este inițializat prin declarare (char s[10]="C.Babbage"), atunci caracterul null este memorat automat în ultima componentă; dacă este inițializat prin citire (ca mai sus), atunci trebuie inclusă în program o instrucțiune explicită de atribuire (s[n]=0;).

B. FACILITĂȚI PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE ÎN LIMBAJUL PASCAL

Limbajul ***Pascal*** oferă un tip de date standard special: tipul **string**. O variabilă de tip **string** poate reține până la 255 caractere și poate fi declarată în două moduri

1.

```
type sir=string;
var  s:sir;
```

în acest caz se rezervă în memorie 256 de octeți – indiferent câte caractere are la citire șirul *s* – primul octet conține lungimea efectivă a șirului citit pentru exemplul șirului "C.Babbage" lungimea șirului este 9) – ;

2.

```
type sir=string[n];
var  s:sir;
```

în acest caz se rezervă în memorie *n* de octeți – primul octet conține lungimea efectivă a șirului citit; – se pot citi șiruri cu maximum *n-1* caractere.

Atenție

Programatorul nu trebuie să se preocupe de depunerea lungimii efective a șirului în primul octet.

Variabila *s* de tipul string va fi privită ca o variabilă independentă dar "componentele ei" vor putea fi accesate prin indexare, la fel ca și componentele oricărui vector!

TEME

1. Ce afișează următorul program?

```
program sir2;
  var s: string;
begin
  writeln('Introduceti sirul: ');
  readln (s);
  writeln('Sirul',s,'are',ord(s[0]),' caractere');
  writeln('s[3]=',s[3],'s[5]=',s[5],'s[6]=',s[6]);
  writeln
end.
```

Operația de atribuire pentru șiruri

2. Variabilele de tip **string** pot fi inițializate – ca orice variabile – prin citire sau atribuire.

Fie variabilele g și k inițializate cu șirurile 'Leibnitz' și respectiv 'Zuse'. Care dintre următoarele moduri de declarare permite efectuarea corectă (fără trunchiere la dreapta) a oricăreia dintre atribuirile: a ← b , b ← a:

```
(1) var a:string;
      b:string;
(4) var a:string;
    b:string[4];

(2) var a:string[4];
      b:string[4];
(5) var a:string[15];
      b:string[13];

(3) var a:string[8];
      b:string[8];
(6) var a:string[20];
      b:string[20];
```

C. FACILITĂȚI PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE ÎN LIMBAJUL C/C++

Limbajul *C/C++* permite citirea și scrierea șirurilor de caractere, fără a mai preciza adresa elementelor. Astfel, primul program care exemplifică prelucrarea șirurilor de caractere devine:

```
#include<iostream.h>
void main (void)
{ char s[50];
  cout<<"Introduceti sirul fara spatii"<<endl;
  cin>>s;
  cout<<"Sirul introdus:
  cout<<s;
}
```

Această metodă prezintă
citită o "frază" în care "cu
Pentru înlăturarea acest
rilor de caractere de ori
Să presupunem că trebu

'John Napier a inventat
stare să reducă efectuar

Acest șir are 182 de
char secv[183];
Prezentăm mai jos câte
(ex1) cin.get(secv,183,'
(ex2) cin.get(secv,183);
(ex3) cin.get(secv,183,'
(ex4) cin.get(secv,86);
În primele două cazuri,
John Napier a inventat nu
funcție predefinită are urmă

Definiție

cin.get(șir, **int** nr, **char**

Semnificația parametril
- primul parametru ac
nterior, de exemplu, prin
- al doilea parametru
mărul de caractere din și
- al treilea parametru a

Efectele apelării funcție
- se citesc nr-1 carac
clusiv spațiile;
- caracterul *null* este in
- atunci când apare, ca

(de aceea în exemplul

```
cin>>s;
cout<<"Sirul introdus:"<<endl;
cout<<s;
}
```

Această metodă prezintă un dezavantaj: șirul citit nu poate conține mai multe spații (nu poate fi citită o "frază" în care "cuvintele" să fie separate prin spații).

Pentru înlăturarea acestui dezavantaj, vom folosi o funcție specială care permite citirea șirurilor de caractere de orice lungime care conțin orice caracter, inclusiv spații.

Să presupunem că trebuie să citim și să prelucrăm următorul – text – șir de caractere:

'John Napier a inventat nu numai logaritmii naturali dar și primul calculator mecanic, în stare să reducă efectuarea înmulțirilor și împărțirilor la efectuarea de adunări și scăderi.'

Acest șir are 182 de caractere, deci poate fi declarat astfel:

char secv[183];

Prezentăm mai jos câteva exemple de citire a acestui șir cu ajutorul funcției **cin.get**.

(ex1) cin.get(secv,183,'.');
(ex2) cin.get(secv,183);
(ex3) cin.get(secv,183,',');
(ex4) cin.get(secv,86);

În primele două cazuri, este citit întregul șir; în ultimele două cazuri este citit numai subșirul *'John Napier a inventat nu numai logaritmii naturali dar și primul calculator mecanic,'*. Această funcție predefinită are următorul antet:

🕮 Definiție

cin.get(șir, **int** nr, **char**='\n')

Semnificația parametrilor:

- primul parametru actual trebuie să fie o variabilă de tip șir de caractere, declarată anterior, de exemplu, prin **char** v[n];
- al doilea parametru actual trebuie să fie un număr cuprins între **0** și **n**, care să indice numărul de caractere din șir care trebuie citite;
- al treilea parametru actual este opțional; dacă nu apare, atunci se subînțelege că este '\n'.

Efectele apelării funcției (în oricare dintre variante) sunt:

- se citesc nr-1 caractere (iată de ce am folosit 183 deși șirul nostru are 182 caractere), inclusiv spațiile;
- caracterul *null* este inserat automat după ultimul caracter din șir;
- atunci când apare, caracterul transmis ca al treilea parametru actual forțează sfârșitul șirului (de aceea în exemplul **(ex3)** de mai sus, citirea se oprește la întâlnirea virgulei).

Atenție

Pentru a citi numai un prefix al șirului, nu întreg șirul, putem folosi:

- varianta **(ex3)**: se citește șirul *secv* până la întâlnirea primului caracter ‘,’;
- varianta **(ex4)**: se citesc primele 85 componente din șirul *secv*.

Pentru a citi un singur caracter, fie că este spațiu sau nu, putem folosi funcția **cin.get** fără parametri: **cin.get();** acest apel este folosit frecvent la citirea succesivă a mai multor șiruri de caracter pentru preluarea separatorului de sfârșit de șir.

D. SUBPROGRAME PREDEFINITE PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE

- **Atribuirea de valori**

LIMBAJUL PASCAL	LIMBAJUL C/C++
operatorul :=	funcția **strcpy**

program atribuire1;
var s1,s2,s3:**string**;
begin
s1:=’Charles’;
s2:=’Babbage’;
writeln(‘Sirurile citite:’);
writeln(s1,’ ‘,s2);
s3:=s2;
s2:=s1;
s1:=s3;
writeln(‘Sirurile dupa interschimbare:’);
writeln(s1,’ ‘,s2);
end.
	#include<iostream.h>
	#include<string.h>
	#include<conio.h>
	void main (void)
	{clrscr();
	char s1[10]=”Charles”, s2[10]=”Babbage”, s3[10];
	cout<<s1<<s2<<**endl**;
	strcpy(s3,s2);
	strcpy(s2,s1);
	strcpy(s1,s3);
	cout<<”Sirurile dupa interschimbare:”<<**endl**;
	cout<<s1<<s2<<**endl**;
	getch();}

	Definiția funcției: **char *strcpy(char** *destinație, **char** *sursa); Apelarea: **strcpy**(s1,s2); Ca urmare, șirul s2 este depus în șirul s1; șirul s2 rămâne nemodificat; șirul s1 va conține rezultatul operației și va fi returnat către programul apelant.
Ambele programe afișează secvențele Charles Babbage și Babbage Charles	

Atenție (C/C++)

Caracterul * care precede numele funcției și al parametrilor formali indică faptul că parametrii nu reprezintă valori ci adrese. Cu alte cuvinte, **sursa* reprezintă adresa primei locații din zona de memorie atribuită acestui parametru al funcției **strcpy**.

- **Compararea șirurilor de caractere**

Șirurilor de caractere li se pot aplica operatorii relaționali (<, >, = etc). Cum putem însă compara litera *a* cu cifra *7* sau cu paranteza deschisă?! Dar șirul 'aX' cu șirul 'Ax'?! Metoda constă în compararea codurilor ASCII prin care se reprezintă în memorie caracterele din șir (așa cum am observat în reprezentarea șirului 'C.Babbage', codul literei a, 097, este predecesorul în sens numeric al codului literei b, 098).

Fie două variabile **x** și **y** de lungime **n**, respectiv **m**, care trebuie comparate. Distingem următoarele cazuri:

- codul primului caracter din **x** este mai mare decât codul primului caracter din **y**; atunci x>y (de exemplu: 'bit' > 'Octet' deoarece *ASCII*('b') = 098 > 079 = *ASCII*('O'));
- codul primului caracter din **x** este mai mic decât codul primului caracter din **y**; atunci x<y (de exemplu: 'Bit' < 'octet' deoarece *ASCII*('B') = 066 < 111 = *ASCII*('o'));
- codul primului caracter din **x** este egal cu codul primului caracter din **y** (este practic același caracter); distingem următoarele subcazuri:
  - n=m=1; atunci x=y;
  - n>m=1; atunci x>y (de exemplu: 'if'>'i');
  - m>n=1; atunci y>x;
  - n≥m>1; atunci se continuă cu compararea codurilor caracterelor de pe pozițiile următoare din **x** și **y**. Să presupunem că pentru toate primele *h* poziții codurile au fost egale:
    - dacă h=m=n atunci x=y;
    - dacă h=m<n atunci x>y;
    - dacă h<m atunci relația dintre **x** și **y** este relația dintre codurile *ASCII* ale caracterelor aflate în **x** și **y** pe poziția h+1.
  - m≥n>1; se procedează analog.

Compararea în Limbajul Pascal	Compararea în Limbajul C/C++
operatorii relaționali	funcția **strcmp**

program comparare1;
var s1,s2:**string**;
begin
writeln('s1:');**readln**(s1);
writeln('s2:');**readln**(s2);
if s1<s2 **then**
writeln(s1,'<',s2)
	#include<iostream.h>
	#include<string.h>
	void main(void)
	{**char** s1[20],s2[20];
	int v;
	cout<<"\n s1:";**cin.get**(s1,20);
	cin.get(); **cout**<<"\n s2:";**cin.get**(s2,20);

else
if(s1>s2) then
writeln(s1,'>',s2)
else
writeln(s1,'=',s2);
end.
	cout<<endl;
	v=strcmp(s1, s2);
	if (v < 0) cout<<s1<<"<"<<s2;
	else
	if (v >0) cout<<s1<<">"<<s2;
	else cout<<s1<<"="<<s2;
	}

	Definiția funcției:
	int strcmp(const char *sir1, const char *sir2);
	Apelarea:
	v=strcmp(s1,s2);
	Funcția compară sirurile s1 și s2 și returnează către programul apelant o valoare întreagă care este:
	< 0 , dacă s1 < s2,
	= 0 , dacă s1 = s2,
	> 0 , dacă s1 > s2.

Pentru s1 = "abc" și s2 = "aac", ambele programe afișează abc > aac	

- **Determinarea lungimii unui șir de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **length** sau valoarea conținută în primul octet din șir	funcția **strlen**
program length1; **var** s1:**string**; **begin** s1:='Acest sir are 27 caractere.'; **writeln**('Acest sir are ',**ord**(s1[0]),' caractere.'); **writeln**('Acest sir are ',**length**(s1),' caractere.'); **end**.	**#include<iostream.h>** **#include<string.h>** **void main (void)** {**char** s1[28]="Acest sir are 27 caractere."; **cout**<<"Acest sir are "<< **strlen**(s1)<<" caractere."; }
Definiția funcției: **function length** (sursa:**string**): :**Integer**; Apelarea: n:= **length**(s); Funcția returnează către programul apelant numărul de caractere din șirul s (primul octet, rezervat automat numărului de caractere din șir, nu este numărat).	Definiția funcției: size_t **strlen** (**char** *sursa); Apelarea: n = **strlen** (s); Funcția returnează către programul apelant numărul de caractere din șirul s (ultimul octet, rezervat automat caracterului null, nu este numărat).
Ambele programe afișează textul „Acest șir are 27 caractere."	

• **Concatenarea şirurilor de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
Funcţia **concat** sau operatorul +	funcţia **strcat**

program concat1;
var s1,s2,r1,r2:**string**;
begin
s1:='Turbo';s2:='Pascal';
r1:=s1+' '+s2;
r2:=**concat**(s1,' ',s2);
writeln('Cu operatorul + : ',r1);
writeln('Cu functia concat: ',r2);
end.
	#include<iostream.h>
	#include<string.h>
	void main (void)
	{**char** s1[10]="Borland",s2[10]="C";
	strcat (s1, " ");
	strcat (s1, s2);
	cout<<"\n"<<s1;
	}

Definiţia funcţiei: **function concat**(s1[,s2,…,sn]:**string**):**string;** Apelarea: r:=**concat**(a1,a2,…,ak); Ca urmare, şirurile a1, a2, …, ak sunt concatenate în această ordine şi de la stânga la dreapta. Dacă şirul rezultat are mai mult de 255 caractere, atunci el este trunchiat la dreapta.	Definiţia funcţiei: **char *strcat** (**char** *destinaţie, **char** *sursa); Apelarea: **strcat** (a1, a2); Ca urmare, şirul a1 este concatenat la dreapta cu şirul a2; şirul s2 rămâne nemodificat; şirul s1 va conţine rezultatul operaţiei şi va fi returnat către programul apelant. Pentru a concatena n şiruri, funcţia trebuie apelată de n-1 ori: **strcat**(a1,a2); **strcat**(a1,a3); . . .; **strcat**(a1,an);
Programul afişează secvenţa „Turbo Pascal" (de două ori!)	Programul afişează secvenţa „Borland C" (o singură dată)

Atenţie

Operaţia de concatenare a şirurilor NU este comutativă.

• **Conversia literelor mici în majuscule**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **upcase**	funcția **strupr**

program upcase1;
var s1:**string**;
i:**integer**;
begin
s1:='Clasa a XI-a';
writeln('Sirul initial: ', s1);
for i:=1 **to** length(s1) **do**
s1[i]:= **upcase**(s1[i]);
writeln('Sirul cu majuscule: ', s1);
end.
	#include<iostream.h>
	#include<string.h>
	void main (void)
	{**char** s1[13]="Clasa a XI-a";
	cout<<"\n Sirul initial: "<<s1<<**endl**;
	strupr (s1);
	cout<<"Sirul cu majuscule: "<<s1;
	}

Definiția funcției: **function upcase** (car: **Char**):**Char**; Apelarea: w:= **upcase** (c); Funcția returnează majuscula corespunzătoare literei c primită ca parametru actual; dacă acest parametru nu se află în domeniul a..z atunci el rămâne neschimbat.	Definiția funcției: **char *strupr(char** *sursa); Apelarea: adr=**strupr** (s1); Funcția returnează șirul s1, în care toate caracterele – aflate în domeniul a..z – au fost transformate în majuscule. Cele din afara acestui domeniu rămân neschimbate.
Ambele programe afișează Clasa a XI-a și, apoi, CLASA A XI-A	

• **Căutarea unui subșir într-un șir de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **pos**	funcția **strstr**

program pos1;
var s1,s2:**string**;
n:**integer**;
begin
s1:='imprimanta';
writeln(s1);
n:=**pos**('prim',s1);
writeln('"prim" incepe din pozitia:',n);
writeln('"rima" incepe din pozitia:',**pos**('rima',s1));
n:=**pos**('prima',s1);
writeln('"prima" incepe din pozitia:',n);
writeln('"manta" incepe din
	#include<iostream.h>
	#include<string.h>
	void main (void)
	{**char** s1[11]="imprimanta",
	s2[5]="prim",s3[5]="rima",
	s4[6]="prima";
	char s5[6]="manta",s7[5]="mira", s6[8]="imprima", *adr;
	adr=**strstr**(s1,s2);
	if (adr) **cout**<<adr-s1+1<<**endl**;
	else cout<<"Nu apare in sir" <<**endl**;
	adr=**strstr**(s1,s3);
	if (adr) **cout**<<adr-s1+1 <<**endl**;

...ția:',pos('manta',s1));
...teln('"imprima" incepe din
...ția:',pos('imprima',s1));
...= pos('mira',s1);
...teln('"mira" incepe din pozitia:',n,',
...a nu se afla in sir');
...
	else cout<<"Nu apare in sir"<<endl;
	adr=strstr(s1,s4);
	if (adr) cout<<adr-s1+1<<endl;
	else cout<<"Nu apare in sir"<<endl;
	adr=strstr(s1, s5);
	if (adr) cout<<adr-s1+1<<endl;
	else cout<<"Nu apare in sir"<<endl;
	adr = strstr(s1, s6);
	if (adr) cout<<adr-s1+1<<endl;
	else cout<<"Nu apare in sir"<<endl;
	adr = strstr(s1, s7);
	if (adr) cout<<adr-s1+1<<endl;
	else cout<<"Nu apare in sir"<<endl;}

...ția funcției: ...ction pos(subsir: String; sir: String): ...; ...area: ...pos(s1,s2); ...ția verifică (de la stânga la dreapta) dacă ...ența s1 apare – în întregime – în ...ența s2. În caz afirmativ, returnează ...icele caracterului din s2 care este și primul ...cter din s1; altfel returnează valoarea 0.	Definiția funcției: **char *strstr(char** *sir, **char** *subsir); Apelarea: adr=**strcstr**(s1,s2); Funcția verifică (de la stânga la dreapta) dacă secvența s2 apare – în întregime – în secvența s1. În caz afirmativ, returnează adresa caracterului din s1 care este și primul caracter din s2; altfel returnează valoarea *null*.
...ramele afișează numerele 3, 4, 3, 6, 1, 0	

- **Conversia unei valori numerice într-o secvență de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
...cedura **str**	funcțiile **itoa**

...gram str1;
...s1:string;
...integer;
...
...=2006;
...:4,s1);
...teln('Numarul ',a:4);
...teln('Sirul ',s1);
...
	#include<iostream.h>
	#include<stdlib.h>
	void main (void)
	{int a=2006;
	char s1[100];
	itoa(a,s1,10);
	cout<<"Numarul:"<<a;
	cout<<"Sirul:"<<s1<<endl;
	}

Definiția procedurii: **procedure str**(numar:i; **var** sursa:**string**); Apelarea: **str**(nr:n,s); Procedura returnează șirul s obținut prin convertirea numărului nr de la tipul întreg la tipul **string**. Parametrul **n** indică numărul total de caractere pe care se face conversia (numărul de octeți ai **s**).	Definiția funcțiilor: **char *itoa(int** valoare, **char** *sir, **int** baza); Apelarea: **itoa**(v1, s1, b1); Funcția **itoa** returnează șirul s1 obținut prin conversia numărului întreg v1 din baza b1 în tipul **char** (dacă b1=10 atunci se reține și semnul lui v1). Există două funcții similare funcției **itoa**:**ltoa** și **ultoa**. prima convertește în șir o valoare de tip **long int**, a doua convertește o valoare de tip **unsigned long.**
Ambele programe afișează secvența "2006".	

- **Conversia unei secvențe de caractere într-o valoare numerică**

LIMBAJUL PASCAL	LIMBAJUL C/C++
procedura **val**	funcțiile **atol, atof**

program val1;
var s1:**string**;
i,c:**integer**;
r:**real**;
begin
s1:='2006';
writeln('Sirul: ',s1);
val(s1,i,c);
if c=0 then **writeln**('Numarul:',i:4)
else
writeln('Eroare: s1= ',s1);
s1:= '-1.23';
writeln('Sirul: ',s1);
val(s1, r, c);
if c=0 **then writeln**('Numarul:',r:5:2)
else
writeln('Eroare: s1= ',s1);
s1:='17 ani';
writeln('Sirul: ',s1);
val(s1,r,c);
if c=0 **then writeln**('Numarul:',r:5:2)
	#include<iostream.h>
	#include<stdlib.h>
	void main (void)
	{**long** l; **float** f;
	char s1[5]="2006";
	char s2[6]="-1.23";
	l=**atol**(s1);
	cout<<"s1: "<<s1<<";l: "<<l<<**endl**;
	l=**atol**("17ani");
	cout<<" l: "<<l<<**endl**;
	f=**atof**(s2);
	cout<<"s2: "<<s2<<"; f: "<<f<<**endl**;
	f=**atof**("3.02*7");
	cout<<" f: "<<f<<**endl**;
	}

else
writeln('Eroare la conversie.');
end.	

Definiția procedurii:
procedure val(sursa; var v:integer; var cod:integer);
Apelarea:
val(s1,nr,c);
Procedura returnează numărul nr (întreg sau real) la care a fost convertit șirul s1 și valoarea întreagă c, indicând modul în care s-a realizat conversia: dacă s1 a putut fi convertit nu a conținut decât cifre, virgulă, punct zecimal, semnul + sau -) atunci c = 0; altfel c ? 0 (conversia nu s-a realizat).
	Definiția funcțiilor:
	long atol(char *sursa);
	double atof(char *sursa);
	Apelarea:
	n = **atol**(s1);
	n = **atof**(s1);
	Funcția **atol** (**atof**) convertește șirul s1 la un întreg de tip **long** (la un real de tip **float**); dacă s1 nu se poate converti, atunci funcția returnează valoarea 0.
	
	Observație
	Există și funcțiile **atoi, strtol, strtoul, strtod** care convertesc un șir la: un întreg, un întreg de tip **long**, un întreg de tip **long** fără semn, respectiv la un real de tip **double**.

Programul afișează - pe 3 rânduri - 2006, -1.23 și Eroare: s1= 17 ani	Programul afișează - pe 4 rânduri - 2006, 17, -1.23 și 3.02

PROBLEME PROPUSE

Realizați, în limbajul de programare studiat, câte un program pentru rezolvarea fiecăreia dintre următoarele probleme:

1. Pentru fiecare elev din clasă se citesc, pe rând, numele și prenumele (scrise cu litere mici și/sau mari).
   Să se afișeze (pentru fiecare elev), pe un singur rând, numele (cu majuscule) și prenumele (doar prima literă să fie majusculă).
   Să se afișeze numele elevului cu cel mai lung prenume și prenumele elevului cu cel mai lung nume.
   Să se ordoneze crescător elevii după numele de familie.
   Să se ordoneze descrescător elevii după prenume.

2. *Să se verifice dacă o frază este de tip palindrom (exemple: Au o nava noua. Icre, pui, ciuperci. Ele fac cafele. Sa nu iei un as. O rama maro.)* În acest scop se vor elimina spațiile (codul ASCII al spațiului este 032).

3. Se citește o "frază". *Se cere să se afișeze:*
   - numărul "cuvintelor" care apar în "frază";
   - cel mai lung și cel mai scurt cuvânt precum și poziția începând cu care apar acestea în "frază";
   - să se șteargă din "frază" primul "cuvânt" care începe cu litera I;
   - să se insereze în "frază", după fiecare "cuvânt", același "cuvânt" dar scris cu majuscule.
4. Se citește o "frază". *Se cere să se afișeze:*
   - *toate perechile de "cuvinte" adiacente care "rimează";*
   - *toate perechile de "cuvinte" neadiacente care "rimează", împreună cu poziția primei litere în frază (considerăm că două cuvinte rimează dacă cel puțin ultimele lor două litere coincid).*
5. Se citește un șir format numai din litere și spații. *Se cere să se afișeze șirul, după ce toate vocalele consecutive au fost înlocuite cu majuscule.*
6. Se citește un șir de cifre zecimale. *Se cere să se afișeze șirul după ce toate cifrele pare p care apar au fost înlocuite cu cifrele corespunzătoare (9–p) iar cifrele impare i cu cifrele corespunzătoare (i-1). Exemplu: pentru șirul 143265 se va afișa șirul 052734.*
7. Statisticile arată că fiecare utilizator de e-mail are 1,8 (!!) căsuțe de poștă electronică. *Să se citească toate adresele e-mail ale unei persoane și să se afișeze separat numele-utilizatorului și adresa serverului (codul ASCII al caracterului @ este 064).*
8. Se citesc mai multe adrese **Web**. *Să se afișeze separat subșirul care reprezintă: schema (http , ftp:// etc.), numele serverului, calea, numele fișierului care trebuie încărcat.*
9. Se citesc două șiruri. *Se cere să se afișeze un subșir comun de lungime maximă.*
10. Se citește un șir. *Se cere să se afișeze cel mai lung subșir format din litere ordonate alfabetic (ab, abbc, afgk etc.).*

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Etapa: *Realizare și testare* (implementarea unităților de program în limbajul de programare studiat)

Cerințe:

1. Realizarea și testarea individuală a programelor.
2. Integrarea programelor într-o aplicație unitară.
3. Completarea documentației cu specificațiile tehnice de programare.

6. RECURSIVITATEA

6.1. DEFINIRE. EXEMPLIFICARE

Toți algoritmii sunt reprezentați printr-o succesiune logică și finită de pași descriși prin structuri specifice: structuri secvențiale, structuri decizionale sau structuri repetitive.

Un algoritm care conține structuri repetitive poate fi reprezentat: *iterativ* sau *recursiv*.

Reprezentarea iterativă a unui algoritm repetitiv implică reluarea parcurgerii secvenței de operații până când este îndeplinită condiția de oprire. În limbajele de programare cunoscute, structurile repetitive sunt implementate iterativ cu ajutorul instrucțiunilor corespunzătoare.

LIMBAJUL PASCAL	LIMBAJUL C/C++

for i:=1 to n do
begin
secvența de operații
end;
for i:=n to downto 1 do
begin
secvența de operații
end;
	for (i=1; i<=n; i++)
	{ secvența de operații }
	
	for (i=n; i>=1; i—)
	{ secvența de operații }

while (cond_logică) do
begin
secvența de operații
end;
	while (cond_logică) do
	{ secvența de operații}

repeat secvența de operații until (cond_logică);	do {}while(cond_logică)

Repetarea unei secvențe de instrucțiuni (S) poate fi determinată și prin organizarea modulară a programului astfel încât S să aparțină unui modul care se autoapelează (cap. 2, fig. 40b). Autoapelul este în acest caz comanda de execuție a secvenței S pentru prelucrarea unor alte date de intrare. Ieșirea din secvența de autoapeluri este controlată printr-o condiție de oprire.

Definirea în matematică a unor relații recurente (șiruri recurente) și prelucrarea acestora au stat la baza construirii mecanismului recursivității pentru reprezentarea algoritmilor.

Exemplu:

Se consideră un număr ***n*** natural nenul. Să se calculeze și să se afișeze n! (factorialul numărului natural *n*).

Varianta iterativă:

Pentru implementarea iterativă se folosește relația:

$$n! = 1*2*...*n \text{ , unde } 0!=1$$

Se apelează din programul principal, respectiv din funcția principală, funcția *fact* care returnează valoarea lui ***n!***. În corpul funcției, variabila ***p*** se inițializează cu valoarea 1 și apoi într-o instrucțiune repetitivă cu contor se repetă de ***n*** ori operația p=p*i, i=1,n. Funcția va returna valoarea finală a produsului ***p***.

IMPLEMENTAREA ITERATIVĂ

LIMBAJUL PASCAL

```
uses crt;
 var n:integer;

function fact(n:integer):longint;
 var i:integer;
    p:longint;
begin
    p:=1;
    for i:=1 to n do
      p:=p*i;
    fact:= p;
end;

begin;
  clrscr;
  write(' n = ');readln(n);
  writeln(' Factorialul ',n,'! = ', fact(n));
end.
```

LIMBAJUL C/C++

```
#include<iostream.h>
 #include<conio.h>

long  fact(int n)
 { int i;
  long p=1;
   for (i=1;i<=n;i++)
     p=p*i;
   return p;
 }

 void main()
 { int n;
  clrscr();
  cout<<" n = ";cin>>n;
  cout<<" Factorialul "<<n<<"! = "<<fact(n);
  }
```

Varianta recursivă:

Se folosește umătoarea relație de recurență:

$$fact(n) = \begin{cases} 1, \text{ pentru } n = 0 \\ n * \text{fact}(n-1) \end{cases}$$

Exemplu: Pentru n=4, atunci n! se determină astfel:

Se apelează din programul principal, respectiv din funcția principală funcția *fact* care returnează valoarea lui ***n!***. Funcția *fact* are ca parametru formal numărul natural ***n***. Funcția

...uită pe baza relației de ...ență, definită în enunțul ...ei. După apel, funcția se ...elează cu parametrul for... (n-1) multiplicat cu *n* adică: *fact(n-1)*. Dacă n>0 atunci se ...ă autoapelul (n → n-1 , iar ...etrul funcției are valoarea ... Șirul de autoapeluri se ... când este îndeplinită *...ția de oprire.*

...form relației de recurență ...torialului deducem *condiția ...prire: n=0.*

...toapelurile au loc de sus în ...apelul principal → condiția ...prire) iar construcția soluției ...ce de jos în sus (condiția de ... → apelul principal), adică:

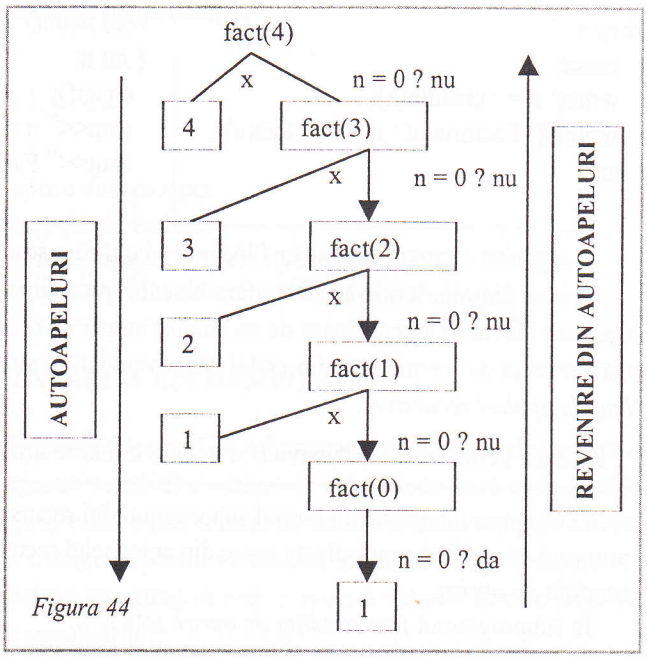


Figura 44

4!= fact(4)=4*fact(3)=4*(3*fact(2))=4*(3*(2*fact(1)))=4*(3*(2*(1*fact(0))))=

Autoapeluri ale subprogramului fact până la întâlnirea condiției de oprire n = 0

= 4*(3*(2*(1*1))) = 4*(3*(2*1))= 4*(3*2)=4*6=24

Construirea soluției după întâlnirea condiției de oprire

IMPLEMENTAREA RECURSIVĂ

LIMBAJUL PASCAL	LIMBAJUL C/C++

...;
...integer;
 	
...fact(n:integer):longint;
...
...=1;
...=0 then fact:=1
else fact:=n*fact(n-1)
...
	#include<iostream.h>
	#include<conio.h>
	
	long fact(int n)
	{ int i;
	if (!n) return 1;
	else
	return n*fact(n-1);
	}

```
begin;
 clrscr;
 write(' n = ');readln(n);
 writeln(' Factorialul ',n,'! = ', fact(n));
end.
```

```
void main()
{ int n;
 clrscr();
 cout<<" n = ";cin>>n;
 cout<<" Factorialul "<<n<<"! = "<<fact(n);
 }
```

algoritm recursiv conține un bloc procedural care se autoapelează, altfel spus se apelează pe sine însuși din interiorul său. Din afara blocului procedural se face un prim apel al acestuia, după care blocul se autoapelează de un anumit număr de ori; la fiecare nouă autoapelare se execută secvența de instrucțiuni din codul său, eventual cu alte date, realizându-se un așa-numit "*lanț de apeluri recursive*".

Pentru a defini corect recursivitatea trebuie urmărite următoarele aspecte:

1. Condiția de oprire: În corpul subprogramului recursiv există unul sau mai multe teste prin care se verifică condițiile de ieșire din autoapelul recusiv. Aceste teste de ieșire formează *condiția de oprire*.

În subprogramul ***fact*** *condiția de oprire* este n=0.

2. Prelucrarea progresivă: Esența definiției recursive constă în rezolvarea cazului curent apelând la cazul precedent. În acest fel, celelalte cazuri ale autoapelului recursiv, diferite de condiția de oprire, trebuie rezolvate prin tratarea lor astfel încât procesul apelului recursiv să le orienteze către una dintre condițiile de oprire.

În exemplul prezentat, valoarea parametrului formal ***n*** este decrementată până când atinge valoarea 0 (*condiția de oprire*).

TEME

1. Stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Un algoritm care nu conține structuri repetitive poate fi reprezentat atât iterativ, cât și recursiv.		
2	Executarea unor secvențe de operații, de un număr cunoscut de ori, este tratată iterativ prin intermediul unor structuri repetive corespunzătoare.		
3	Executarea unor secvențe de operații, de un număr cunoscut de ori, este tratată recursiv prin intermediul autoapelurilor aceluiași subalgoritm de un număr finit de ori.		
4	Subalgoritmul se autoapelează până când este îndeplinită condiția de oprire.		

Se consideră următoarea definiție (relație de recurență):

$$f(n) = \begin{cases} 1, \text{ dacă } n = 1 \\ \dfrac{f(n-1)}{a + f(n-1)}, \text{ dacă } n > 1,\ a \in R^* \end{cases}$$

a) Stabiliți care este condiția de oprire din autoapel.

b) Pentru n=4 construiți toate autoapelurile realizate.

Analizați următorul enunț și puneți în evidență aspectele recurente:

Predecesorul oricărui număr natural este un număr natural: zero este număr natural.

MECANISMUL DE IMPLEMENTARE A RECURSIVITĂȚII

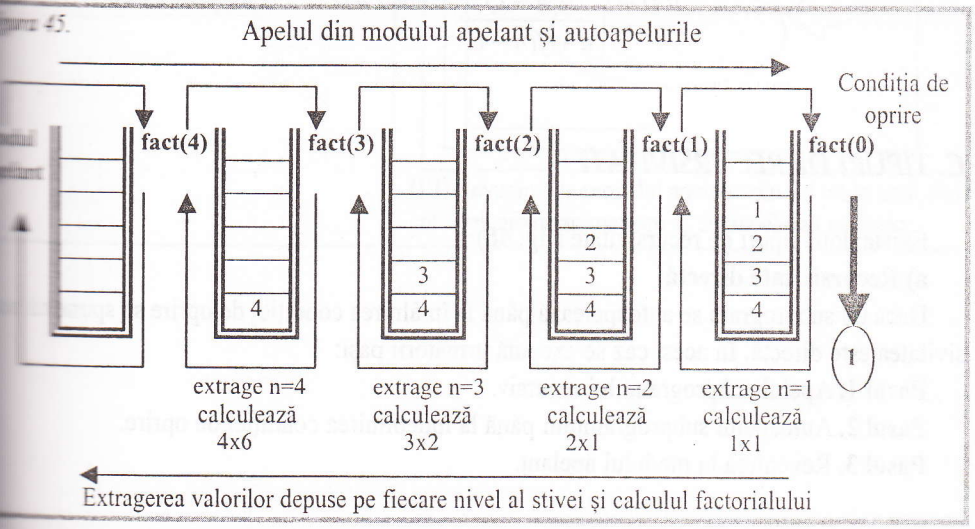
În capitolul *'Subprograme'* s-a precizat că orice apel de subprogram are ca efect salvarea într-o zonă de memorie denumită stivă (segment de stivă) a adresei de revenire de după apel, a valorilor parametrilor formali transmiși prin valoare și a adresei parametrilor transmiși prin adresă, precum și alocarea de spațiu pe stiva sistemului pentru variabilele locale ale subprogramului.

La revenirea în subprogramul apelant, se extrag în ordine inversă valorile salvate pe stivă (mai întâi ale variabilelor locale, cât și ale parametrilor).

În cazul subprogramelor recursive, la apelul subprogramului se depun pe primul nivel al stivei sistemului, în ordine, parametrii formali transmiși, se alocă spațiu pentru variabilele locale (dacă este cazul) și adresa de revenire după apel. La fiecare autoapel al subprogramului, se urcă cu câte un nivel în stivă, pe acest nivel se depun noile valori ale parametrilor transmiși, se alocă spațiu pentru variabilele locale. Când se întâlnește condiția de oprire se încheie autoapelul subprogramului, se extrag din stivă (prin vârful stivei, de sus în jos) în ordine inversă valorile depuse și se efectuează prelucrările impuse în secvența de operații executată la fiecare autoapel. Când se revine în modulul apelant, stiva sistemului este vidă.

Exemplificarea mecanismului recursivității pentru determinarea valorii funcției 4!=fact(4).

Figura 45.



TEME

1. Stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	Afirmații	Răspuns
		A
1	Prin apelul unui subprogram se salvează în *segmentul de date* adresa de revenire de după apel, valorile parametrilor formali transmiși prin valoare și adresele parametrilor transmiși prin adresă și variabilele locale (dacă este cazul) ale subprogramului.	
2	La fiecare autoapel al subprogramului se urcă cu câte un nivel în stivă pe care se depun noile valori ale parametrilor formali și se alocă spațiu pentru variabilele locale.	
3	Când se întâlnește condiția de oprire se încheie autoapelul subprogramului, se extrage din stiva sistemului în *ordinea depunerii* lanțul de valori depuse și se efectuează prelucrările impuse.	
4	În modulul apelant se revine din subprogramul recursiv când s-au extras toate valorile depuse în stiva sistemului (stiva este vidă).	

2. Reprezentați conținutul stivei pe durata executării apelului principal, al autoapelurilor, extragerii valorilor din stivă, construirea valorii returnate și revenirea în subprogramul apelant pentru următoarea definiție (relație de recurență):

$$f(n)=\begin{cases}1, \text{ dacă } n=1\\ \dfrac{f(n-1)}{a+f(n-1)}, \text{ dacă } n>1,\ a\in R^*\end{cases}$$

C. TIPURI DE RECURSIVITATE

Există două tipuri de recursivitate (fig. 46):

a) Recursivitate directă

Dacă un subprogram se autoapelează până la întâlnirea condiției de oprire se spune că recursivitatea este directă. În acest caz se execută următorii pași:

Pasul 1. Apelul subprogramului recursiv.

Pasul 2. Autoapelul subprogramului până la îndeplinirea condiției de oprire.

Pasul 3. Revenirea la modulul apelant.

e, prin marcarea căsuței co

	Răspun	
	A	F
e date adresa de mișii prin valoare locale (dacă este		
nivel în stivă pe ocă spațiu pentru		
subprogramului, de valori depuse		
când s-au extras		

ui principal, al autoapelu
și revenirea în subprogr

$\in R^*$

ției de oprire se spune că

dției de oprire.

b) Recursivitate indirectă

Dacă există două sau mai multe subprograme care se apelează reciproc până când se întâlnește condiția de oprire în cel puțin unul dintre ele, recursivitatea este indirectă sau încrucișată. În acest caz se execută următorii pași:

Pasul 1. Apelul subprogramului.
Pasul 2. Apeluri încrucișate până la întâlnirea condiției de oprire.
Pasul 3. Revenirea la modulul apelant.

Figura 46

Reprezentarea recursivității directe	Reprezentarea recursivității indirecte
(II) Autoapelul (I) Apelul condiția de oprire (III) Revenire la modulul apelant	(I) Apelul subprogramului (II) apel 1-2 (II) apel 1-3 (II) apel 2-3 (III) Revenirea la modulul apelant din cel puțin unul dintre cele trei subprograme care se autoapelează reciproc

Implementarea recursivității indirecte în limbajele de programare are anumite particular

LIMBAJUL PASCAL	LIMBAJUL C/C ++
În limbajul Pascal se poate apela un subprogram care nu a fost declarat, folosind o declarație fictivă (convențională) care este formată din **antet** (numele subprogramului urmat eventual de lista parametrilor formali) și cuvântul rezervat **FORWARD**, care este separate de antet prin ';'. Această declarație de funcție sau procedură permite referirea la ea fără a fi definită în momentul apelării. Corpul subprogramului se va scrie fiind precedat din nou de antetul acestuia fără a mai fi necesar declararea parametrilor formali.	În limbajul C/C++ se poate apela o funct care nu a fost declarată, folosind o declara fictivă (convențională) care este formată d **antet** (numele subprogramului urmat event de lista parametrilor formali). Această decl rație de funcție permite referirea la ea fără definită în momentul apelării. Corpul funct se va scrie fiind precedat din nou de ante acesteia.

Exemplu *de implementare a recursivității indirecte:*

Să se afișeze toate numerele naturale cuprinse în intervalul dat [a, b].

De pe prima linie a fișierului text ***rec_ind.in*** se citesc numerele naturale **a** și **b** (a<b) sep rate prin câte un spațiu. Numerele naturale din intervalul [a, b] sunt afișate în fișierului ***rec_ind.out***, separate prin câte un spațiu.

Rezolvare:

Din programul principal, respectiv din funcția principală se apelează subprogramul *ment(a)* cu parametrul efectiv ***a***. Acest subprogram are rolul de a genera toate numerele natu consecutive, începând cu ***a***. Din acest subprogram se apelează subprogramul ***scrie(n)***, car rolul de a scrie în fișerul ***rec_ind.out*** valoarea parametrului formal ***n*** transmis și apoi apel subprogramul ***increment(a)*** până când este îndeplinită condiția de oprire (n<b).

Implementarea problemei:

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a,b:integer;
f,g:text;
 	
procedure increment(n:integer); forward;
 	
procedure scrie(n:integer);
begin
write(g,n,' ');
if(n<b) then
increment(n+1);
end;
	#include<fstream.h>
	
	ifstream f ("rec_ind.in");
	ofstream g ("rec_ind.out");
	
	int a,b;
	
	void increment(int n);
	
	void scrie(int n)
	{ g<<n<<" ";
	if(n<b)

procedure increment(n:integer);
begin
scrie(n);
end;

begin
assign(f,'rec_ind.in');reset(f);
assign(g,'rec_ind.out');rewrite
read(f,a,b);
increment(a);
close(f);close(g);
end.

Rec_ind.in
12 20

APLICAȚII IMPLEMENT

Recomandări pentru elabor

1. Înainte de a trece la implem corect relația recursivă; pot
   - relația recursivă este o fo
   - relația de recurență trebu
2. Necesitatea existenței în sub se evita autoapelarea la inf
3. *Reducerea listei parametril* vități dezavantajoase. Pot f trebuie stocate în stivă, de
4. Greșeli frecvente în elabora
   - dacă subprogramul recu referință (prin adresă) și temului se va umple fo chiar și pentru un numă
   - un autoapel incorect co
   - condiția de ieșire din a stivei sistem (mesajul: S

LIMBAJUL PASCAL	LIMBAJUL C/C++

procedure increment(n:integer);
begin
scrie(n);
end;
 	
begin
assign(f,'rec_ind.in');reset(f);
assign(g,'rec_ind.out');rewrite(g);
read(f,a,b);
increment(a);
close(f);close(g);
end.
	increment(n+1);
	}
	
	void increment(int n)
	{
	scrie(n);
	}
	
	void main()
	{f>>a>>b;
	increment(a);
	f.close();g.close();
	}

Rec_ind.in 12 20	***Rec_ind.out*** 12 13 14 15 16 17 18 19 20

2. APLICAȚII IMPLEMENTATE RECURSIV

Recomandări pentru elaborarea unui subprogram recursiv:

1. Înainte de a trece la implementarea unui subprogram recursiv este necesar să se identifice corect relația recursivă; pot exista două situații:
   - relația recursivă este o formulă de recurență matematică;
   - relația de recurență trebuie determinată din enunțul problemei.
2. Necesitatea existenței în subprogram a unei condiții corecte de oprire din autoapeluri pentru a se evita autoapelarea la infinit.
3. *Reducerea listei parametrilor formali ai subprogramului* poate elimina riscul unei recursivități dezavantajoase. Pot fi eliminate din lista parametrilor formali acele argumente care nu trebuie stocate în stivă, declarându-le ca variabile globale.
4. Greșeli frecvente în elaborarea unui program recursiv:
   - dacă subprogramul recursiv conține parametrii formali transmiși prin valoare sau prin referință (prin adresă) și/sau variabile locale care ocupă spațiu mare de memorie, stiva sistemului se va umple foarte repede, ajungându-se la depășirea spațiului rezervat acesteia chiar și pentru un număr mic de autoapeluri;
   - un autoapel incorect conduce la un rezultat eronat;
   - condiția de ieșire din autoapel greșit formulată conduce la autoapel infinit și umplerea stivei sistem (mesajul: *Stack overflow)*.

Concluzie:

Pentru anumiți algoritmi este posibil să se transforme reprezentarea recursivă într-o reprezentare simplă, iterativă.

✓ *Recursivitatea reduce complexitatea unor algoritmi.*

✓ *Deși programele recursive sunt mai dificil de elaborat, totuși ele sunt foarte puternice, formând baza limbajelor inteligenței artificiale.*

✓ *La alegerea între metoda recursivă și iterativă în elaborarea unui program, trebuie să se țină seama de eficiența oferită programului de către fiecare dintre variante, urmărind timpul de rulare și spațiul de memorie necesar. Nu în ultimul rând, trebuie urmărită claritatea programului.*

1. ȘIRUL LUI FIBONACCI

Să se afișeze al ***n***-lea termen al șirului lui Fibonacci, folosind un program care să respecte următoarea definiție:

$$fibo(n) = \begin{cases} 1, \text{ dacă } n < 2 \\ \text{fibo}(n-1) + \text{fibo}(n-2), \text{ altfel} \end{cases}$$

Soluție:

Se scrie o funcție recursivă *fibo(n)*, care va returna al ***n***-lea termen al șirului. Conform definiției, acesta este egal cu predecesorul său f_{n-1} pe care l-ar returna apelul *fibo(n-1)* plus anteprecedentul său f_{n-2} pe care l-ar returna apelul *fibo(n-2)*. Deci în interiorul apelului fibo(n) apar două apeluri recursive. Condiția de oprire din lanțul de apeluri recursive este „*n=1* sau *n=2*".

LIMBAJUL PASCAL

```
var n:integer;

function fibo(n:integer):integer;
 begin
  if n>1 then fibo:=fibo(n-1)+fibo(n-2)
         else fibo :=1;
 end;

 begin;
  write(' n = ');readln(n);
  writeln(' Elementul ',n,' = ', fibo(n));
  readln
 end.
```

LIMBAJUL C/C++

```
#include<iostream.h>

int fibo(int n)
 {
  if (n>1)
     return (fibo(n-1)+fibo(n-2));
  else
     return 1;
 }
 void main()
  { int n;
    cout<<" n = "; cin>>n;
    cout<<" Elementul "<<n<<" = "<<fibo(n);
 }
```

2. CEL MAI MARE DIVIZOR COMUN

Dându-se două numere întregi ***a*** și ***b*** să se determine cel mai mare divizor comun al celor două numere, folosind:

a) algoritmul lui Euclid (se aplică algoritmul lui Euclid, cunoscut la matematică);

b) algoritmul lui Nicomachus, descris de următoarea relație:

$$cmmdc(a,b) = \begin{cases} a, \text{ dacă } a = b \\ cmmdc((a-b),b), \text{ dacă } a > b \\ cmmdc(a, (b-a)), \text{ în rest} \end{cases}$$

a) Algoritmul lui Euclid

Se construiește funcția recursivă *cmmdc(a,b)* care va returna valoarea ultimului rest diferit de zero, respectiv *b* sau *r*. Algoritmul prelucrează perechi de date întregi, de orice semn și cu orice relație de ordine între ele. Se observă că la fiecare autoapel funcția va executa aceeași secvență de instrucțiuni (un "pas" al algoritmului), dar cu alte valori ale lui ***a*** și ***b***. Oprirea lanțului de apeluri recursive are loc când restul împărțirii întregi al celor doi parametri este zero.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a,b:integer;
 	
function cmmdc(a,b:integer):integer;
begin
if a mod b <>0 then
cmmdc:=cmmdc(b, a mod b)
else cmmdc:=b;
end;
 	
begin;
write(' a = ');readln(a);
write(' b = ');readln(b);
writeln(' (',a,' , ',b,') = ', cmmdc(a,b));
end.
	#include<iostream.h>
	
	int cmmdc(int a, int b)
	{
	if ((a%b)!= 0)
	return (cmmdc(b,a%b));
	else return b;
	}
	
	void main()
	{ int a,b;
	cout<<" a = ";cin>>a;
	cout<<" b = ";cin>>b;
	cout<<" ("<<a<<", "<<b<<") = "<<cmmdc(a,b);
	}

b) Algoritmul lui Nicomachus

Se construiește funcția recursivă *cmmdc(a,b),* la fiecare autoapel funcția va executa aceeași secvență de instrucțiuni (un "pas" al algoritmului), dar cu alte valori ale lui ***a*** și ***b***. Oprirea lanțului de apeluri recursive are loc când cei doi parametri au devenit egali. Pe baza acestor observații, definiția recursivă a algoritmului este evidentă, iar funcția recursivă transpune efectiv într-o instrucțiune ***if*** această definiție. Algoritmul prelucrează perechi de date întregi, de orice semn și cu orice relație de ordine între ele, dar la apelul din programul principal sau din funcția principală parametrii efectivi trebuie să fie în valoare absolută.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var a,b:integer;
 	
function cmmdc(a,b:integer):integer;
 	
begin
if (a = b) then cmmdc:=a
else if (a>b) then cmmdc:= cmmdc(a-b,b)
else cmmdc:=cmmdc(a,b-a);
end;
 	
begin;
clrscr;
write(' a = ');readln(a);
write(' b = ');readln(b);
writeln(' (',a, ',',b,') = ', cmmdc(abs(a),abs(b)));
end.
	#include<iostream.h>
	#include<math.h>
	
	int cmmdc(int a, int b)
	{
	if (a==b) return a;
	else if (a>b) return (cmmdc(a-b,b));
	else return (cmmdc(a,b-a));
	}
	void main()
	{ int a,b;
	cout<<" a = ";cin>>a;
	cout<<" b = ";cin>>b;
	cout<<" ("<<a<<","<<b<<") = ";
	cout<<cmmdc(abs(a),abs(b));
	}

3. CITIREA ELEMENTELOR UNUI TABLOU UNIDIMENSIONAL

Să se scrie un program pentru citirea și afișarea elementelor unui șir ***y*** cu ***m*** elemente reale. Dimensiunea șirului și elementele acestuia se citesc de la tastatură, iar afișarea se va face pe monitor.

Rezolvare:

Pentru citirea elementelor șirului de numere se utilizează subprogramul recursiv citire care are ca paramnetru indicele primului element citit. Acest indice este incrementat la fiecare autoapel, până când devine egal cu dimensiunea **n** a șirului. Condiția de oprire este i=n. În mod asemănător se construiește și subprogramul recursiv de afișare a elementelor tabloului.

LIMBAJUL PASCAL	LIMBAJUL C/C++

var y:array[1..20] of real; m: integer;
procedure citeste(i : integer);
begin
if(i<=m) then
begin
read(y[i]);
citire(i+1)
end;
end;
 	
begin
read(m) ; citeste(1);
end.
	#include<iostream.h>
	float y[20];
	int m;
	void citire(int i)
	{
	if(i<=m)
	{ cin>>y[i];
	citire(i+1);
	}
	
	void main()
	{ cin>>m;
	citire(1); }

APLICAȚII DE LABORATOR

1. Scrieți un program pentru completarea elementelor unui șir ***y*** de numere naturale și afișarea acestora, utilizând subprograme recursive. Numărul de elemente ***t*** este citit de la tastură, fiecare element al șirului este generat aleatoriu (0<y[i]< 80, i=1,***t***). Elementele șirului se vor afișa pe monitor (separate prin câte un spațiu).

2. Stabiliți care este efectul programului următor. Formulați un enunț al unei aplicații care se poate rezolva cu ajutorul acestui program.

LIMBAJUL PASCAL

```
var a:array[1..20,1..20] of integer;
  m,n:integer;
  f,g:text;
procedure citire(i,m,j,n:integer);
 begin
  if(i<=m) then
  if(j<=n) then
   begin
   read(f,a[i][j]); citire(i,m,j+1,n);
   end
  else
    citire(i+1,m,1,n);
end;
procedure afisare(i,m,j,n>integer);
 begin
  if(i<=m) then
  if(j<=n) then
    begin
    write(g,a[i][j],' '); afisare(i,m,j+1,n);
    end;
end;
begin
 assign(f,'cit_mat.in');reset(f);
 assign(g,'cit_mat.out');rewrite(g);
 read(f,m,n);      citire(1,m,1,n);
 afisare(1,m,1,n);      close(f);close(g);
end.
```

LIMBAJUL C/C++

```
#include<fstream.h>
ifstream f("cit_mat.in");
ofstream g("cit_mat.out");
int a[20][20];
void citire(int i,int m,int j,int n)
{
  if(i<=m)
  if(j<=n)
     { f>>a[i][j];
       citire(i,m,j+1,n);
     }
     else
       citire(i+1,m,1,n);
}
void afisare(int i, int m, int j, int n)
{
if(i<=m)
if(j<=n)
   { g<<a[i][j]<<" ";      afisare(i,m,j,+1,n);
   }
else
   { g<<endl;      afisare(i+1,m,1,n);
}
void main()
 { int m,n;
   f>>m>>n;
   citire(1,m,1,n);
   afisare(1,m,1,n);
   g.close();
   f.close();
}
```

3. Din fișierul ***matrice.in*** se citește o matrice ***a*** cu ***m x n*** elemente întregi, astfel: de pe prima linie a fișierului se citesc numerele naturale nenule ***m*** și ***n*** separate prin câte un spațiu, iar de pe următoarele ***m*** linii se citesc ***n*** numere întregi separate prin câte un spațiu.

Datele de ieșire se vor afișa, pe linii distincte, în fișierul ***matrice.out.***

Să se scrie un program pentru :

a) citirea elementelor matricei din fișierul ***matrice.in***

b) afișarea elementelor matricei în fișierul ***matrice.out***

4. SUMA ELEMENTELOR UNUI ȘIR

Se consideră un șir ***x*** cu ***n*** numere reale. Să se calculeze suma celor **n** elemente ale tabloului.

Datele de intrare se citesc din fișierul ***suma.in***, astfel: de pe prima linie se citește dimensiunea ***n*** a tabloului numeric ($n \in N$, $1<n<100$), iar de pe următoarea linie se citesc ***n*** numere reale separate prin câte un spațiu.

În fișierul ***suma.out*** se va afișa șirul dat, pe prima linie, iar următoarea linie un număr real cu trei zecimale, reprezentând suma elementelor.

LIMBAJUL PASCAL

```
var x:array[1..20] of real;      n:integer;
   f,g:text;
procedure citire(i:integer);
begin
 if(i<=n) then
   begin
     read(f,x[i]);       citire(i+1);
   end;
end;
procedure afisare(i:integer);
begin
 if(i<=n) then
   begin
     write(g,x[i]:2:3,' ');       afisare(i+1);
   end;
end;
function suma(i:integer):real;
 begin
 if(i<=n) then
    suma:= x[i]+suma(i+1)
  else
     suma:=0;
end;
begin
 assign(f,'suma.in');reset(f);
 assign(g,'suma.out');rewrite(g);
 read(f,n);  citire(1);  afisare(1); writeln(g);
 writeln(g,' suma = ', suma(1):2:3);
 close(f); close(g);
end.
```

LIMBAJUL C/C++

```
#include<fstream.h>
#include<iomanip.h>
ifstream f("suma.in");
ofstream g("suma.out");
float x[20];    int n;
void citire(int i)
{
 if(i<=n)
   { f>>x[i];       citire(i+1);    }
}
void afisare(int i)
{
 if(i<=n)
   { g<<setprecision(3)<<x[i]<<" ";
     afisare(i+1);      }
}
float suma(int i)
{
 if(i<=n)
    return x[i]+suma(i+1);
  else
   return 0;
}
void main()
{ f>>n;
 citire(1); afisare(1);
 g<<" suma = "<<setprecision(3)<<suma(1);
 g.close();f.close();
}
```

APLICAȚIE DE LABORATOR

Se consideră un șir ***x*** cu ***n*** elemente întregi. Datele de intrare se citesc din fișierul ***min_max.in*** astfel: de pe prima linie un număr ***n*** natural nenul (care reprezintă numărul de elemente din șir), iar de pe următoarea linie ***n*** valori întregi separate prin câte un spațiu (care reprezintă elementele șirului **x**) .

Să se se scrie câte un program pentru fiecare cerință :

a) Determinarea simultană (în același subprogram) a minimului și a maximului.

Maximul și minimul se vor afișa în fișierul ***min_max.out***.

Exemplu: ***min_max.in*** ***min_max.out***

5 Minimul = -3

6 2 1 7 -3 Maximul = 7

b) Modificați programul construit la punctul a) (salvați-l cu un alt nume) astfel încât să determinați simultan (în același subprogram) minimul și poziția acestuia în șir. Minimul și poziția acestuia se vor afișa în fișierul ***min_poz.out***.

Exemplu: ***min_max.in*** ***min_poz.out***

5 Minimul = -3

6 2 1 7 -3 Poziția = 5

c) Modificați programul construit la punctul a) sau b) (salvați-l cu un alt nume) astfel încât să determinați simultan (în același subprogram) minimul, poziția minimului în șir, maximul și poziția maximului în șir. Minimul, poziția minimului, maximului și poziția maximului în șir se vor afișa în fișierul ***min_max3.out***.

Exemplu: ***min_max.in*** ***mip_map.out***

5 Minimul = -3 pe poziția 5

6 2 1 7 -3 Maximul = 7 pe poziția 4

5. PRODUSUL SCALAR

Se consideră două șiruri de ***n*** numere întregi ***x*** și ***y***, având amândouă același număr de elemente ***n***, să se calculeze produsul lor scalar.

Produsul scalar se definește ca fiind un număr rezultat prin însumarea produselor elementelor cu același indici din cele două tablouri.

Pentru datele problemei, produsul scalar = $x_1y_1 + x_2y_2 + \ldots + x_ny_n$.

Datele de intrare se citesc din fișierul ***scalar.in*** astfel: de pe prima linie un număr natural nenul ***n*** (dimensiunea celor două tablouri), de pe următoarea linie câte ***n*** valori întregi, separate prin câte un spațiu reprezentând elementele tabloului ***x***, iar de pe ultima linie ***n*** valori întregi, separate prin câte un spațiu reprezentând elementele tabloului ***y***. Valoarea produsului scalar se va afișa pe monitor.

Observație: Problema are scopul de a exemplifica un proces recursiv în care participă mai multe tipuri de date.

Rezolvare:

Funcția recursivă primește ca parametru indicele curent al elementelor din cele două tablouri, al căror produs este depus pe nivelul curent al stivei sistem. Condiția de oprire este îndeplinită atunci când indicele este egal cu *n* . Tablourile au fost declarate global.

LIMBAJUL PASCAL	LIMBAJUL C/C++

type vect=array[1..20]of integer;
 	
var n:integer;
x,y: vect;
f:text ;
 	
function prod_sc(i:integer):integer;
begin
if (i<=n) then
prod_sc:= x[i]*y[i] + prod_sc(i+1)
else prod_sc:= 0 ;
end;
 	
procedure citeste(n : integer; var x:vect);
begin
for i:=1 to n do
read(f, x[i])
end;
end;
begin
assign('scalar.in'); reset(f); read(f,n);
citeste(n,x); citeste(n,y);
write(' produsul scalar = ', prod_sc(1));
close(f)
end.
	include<fstream.h>
	ifstream f("scalar.in");
	int n, vect x[20] ,y[20];
	
	int prod_sc(int i)
	{
	if (i<=n)
	return (x[i]*y[i] + prod_sc(i+1));
	else return 0 ;
	}
	
	void citeste(int n, vect x)
	{int i;
	for (i=1;i<=n;i++)
	f>>x[i];
	}
	
	void main()
	{
	f>>n;
	citeste(n,x);
	citeste(n,y);
	cout<<" Produsul scalar = "<<prod_sc(1);
	f.close();
	}

APLICAȚII DE LABORATOR

1. Se consideră un șir ***x*** cu ***n*** (0<n<10) elemente reale. Datele de intrare se citesc din fișierul ***operatii.in*** astfel: de pe prima linie un număr ***n*** natural nenul (care reprezintă numărul de elemente din șir), iar de pe următoarea linie ***n*** valori întregi separate prin câte un spațiu (care reprezintă elementele șirului **x**).

Datele de ieșire se vor afișa, pe linii distincte, în fișierul ***operatii.out.***

Să se scrie un program:
a) citirea elementelor șirului din
b) afișarea elementelor șirului în
c) determinarea și afișarea sumei
d) determinarea și afișarea produ
e) determinarea și afișarea numă
zecimale);
f) determinarea și afișarea medie
patru zecimale).

Exemplu:

operatii.in
5 2 3 154 23 2 4

2. Se consideră o matrice $A_{n,n}$ cu
matrice.in astfel: de pe prima li
n linii se citesc **n** numere întregi
fișierul ***matrice.out***. Cerințe:
a) Citirea elementelo
b) Afișarea elemente
c) Determinarea și af
d) Determinarea și af
e) Afișarea elemente
f) Afișarea elemente
g) După interschimb
Pentru fiecare cerință scrieți câ

Exemplu:

matrice.in	
4 2 3 1 5 4 6 0 7 9 5 8 1 3 4 1 0	2 3 1 5 4 6 0 7 9 5 8 1 3 4 1 0 Suma Dp= 16 Produsul Ds= 0

Să se scrie un program:

a) citirea elementelor șirului din fișierul ***operatii.in***

b) afișarea elementelor șirului în fișierul ***operatii.out***

c) determinarea și afișarea sumei elementelor șirului (cu exact trei zecimale);

d) determinarea și afișarea produsului elementelor șirului (cu exact trei zecimale);

e) determinarea și afișarea numărului de elemente strict negative din șir (cu exact două zecimale);

f) determinarea și afișarea mediei aritmetice a elementelor strict pozitive din șir (cu exact patru zecimale).

Exemplu:

operatii.in	***operatii.out***
6 2 3 154 23 2 4	3 154 23 2 4 Suma = 198 Produsul = 1020096 Numarul de elemente impare = 2 Suma ultimelor cifre = 18

2. Se consideră o matrice $A_{n,n}$ cu elemente întregi. Datele de intrare se citesc din fișierul ***matrice.in*** astfel: de pe prima linie se citește numărul natural ***n*** nenul, iar de pe următoarele **n** linii se citesc **n** numere întregi separate prin câte un spațiu. Datele de ieșire se vor afișa în fișierul ***matrice.out***. Cerințe:

a) Citirea elementelor matricei din fișier.

b) Afișarea elementelor matricei în fișierul de ieșire.

c) Determinarea și afișarea sumei elementelor de pe diagonala principală.

d) Determinarea și afișarea produsului elementelor de pe diagonala secundară.

e) Afișarea elementelor situate deasupra diagonalei principale.

f) Afișarea elementelor situate sub diagonala secundară.

g) După interschimbarea liniei 1 cu linia n să se afișeze noua matrice.

Pentru fiecare cerință scrieți câte un subprogram recursiv.

Exemplu:

matrice.in				
4 2 3 1 5 4 6 0 7 9 5 8 1 3 4 1 0	2 3 1 5 4 6 0 7 9 5 8 1 3 4 1 0 Suma Dp= 16 Produsul Ds= 0	Elementele situate deasupra diagonalei principale 3 1 5 0 7 1	Elementele situate sub diagonala secundară 7 8 1 4 1 0	Matricea după interschimbare 3 4 1 0 4 6 0 7 9 5 8 1 2 3 1 5

6. ORDONAREA ELEMENTELOR UNUI ȘIR

Se consideră un șir **x** cu ***n*** numere întregi. Să se ordoneze crescător elementele șirului.

Dimensiunea șirului și cele ***n*** elemente ale sale se citesc de la tastatură, iar șirul ordonat se afișează pe ecran.

Rezolvare:

Subprogramul recursiv *ordon(a,n)* va returna șirul ordonat crescător, prin metoda interschimbării. Autoapelurile se încheie când este îndeplinită condiția de oprire *i=n*.

LIMBAJUL PASCAL

```
type vect=array[1..20]of integer;

var n,i:integer;
   a:vect;

procedure ordon(var x:vect; n,i:integer)
 var aux:integer;
  begin
    if i<n then
          if x[i]>x[i+1] then
                          begin
                          aux:=x[i];
                           x[i]:=x[i+1];
                           x[i+1]:=aux;
                           ordon(x,n,1)
                          end
                        else ordon(x,n,i+1)
 end;

procedure citeste(n,x);
 begin
  for i:=1 to n do
   begin
    write(' elementul ',i,' = '); readln(x[i])
   end;
 end;

begin;
 write(' n = ');readln(n);
 writeln(' Elementele sirului a ');writeln;
 citeste(n,a);writeln;
 ordon(a,n,1);
 writeln(' sirul ordonat crescator : ');
 for i:=1 to n do write(a[i],' ')
end.
```

LIMBAJUL C/C++

```
#include<iostream.h>

int vect[20],n;

void  ordon(int x,int n, int i)
 { int aux;
  if(i<n)
   if (x[i]>x[i+1])
      {  aux=x[i];
        x[i]=x[i+1];
        x[i+1]=aux;
        ordon(x,n,1);
      }
   else ordon(x,n,i+1);
 }

void citeste( int n, int x)
 {int i;
 for (i=1;i<=n;i++)
  {
   cout<<" elementul "<<i<<" ";
   cin>>x[i];
  }
 }

void main()
 { cout<<" n = ";cin>>n;
   cout<<" citirea sirului a "<<endl;
  citeste(n,a);
  cout<<endl;
  ordon(a,n,1);
  cout<<" Sirul ordonat = "<<endl;
  for(i=1;i<=n;i++) cout<<a[i]<<" ";
 }
```


7. SCRIEREA UNUI NUM

Se citește un număr natural

Rezolvare:

Subprogramul recursiv *baza*

și câtul împărțirii lui **n** la **b** (c

ultimul cât este 0).

LIMBAJUL PASCAL

```
uses crt;
var n,b:integer

procedure  baza(n:integer; b:i
 var r:integer;
 begin
  r:=n mod b;
  if (n>=b)
    baza(n div b,b);
  write(r);
 end;

 begin
  clrscr;
  write(' n = ');readln(n);
  repeat
   write(' baza = ');readln(b)
  until(b<2 and b>10 );
  write(n,'→', baza(n,b);
 end.
```

8. INVERSAREA UNUI C

Să se inverseze caracterele

caracterului *spațiu*.

Rezolvare:

Se scrie un subprogram fără

un nou caracter ***c***. Condiția de

este *spațiu*. Afișarea caractere

(*write*/*cout*). Când este îndepl

autoapel sunt extrase și afișate

7. SCRIEREA UNUI NUMĂR ÎNTR-O BAZĂ

Se citește un număr natural nenul ***n***. Să se transforme în baza ***b***, unde b=2..9.

Rezolvare:

Subprogramul recursiv *baza(n)* memorează pe fiecare nivel al stivei restul împărțirii lui **n** la **b** și câtul împărțirii lui **n** la **b** (care reprezintă noua valoare a lui **n**). Condiția de oprire este n=0 (ultimul cât este 0).

LIMBAJUL PASCAL	LIMBAJUL C/C++

uses crt;
var n,b:integer
 	
procedure baza(n:integer; b:integer);
var r:integer;
begin
r:=n mod b;
if (n>=b)
baza(n div b,b);
write(r);
end;
 	
begin
clrscr;
write(' n = ');readln(n);
repeat
write(' baza = ');readln(b);
until(b<2 and b>10);
write(n,'→', baza(n,b);
end.
	#include<iostream.h>
	#include<conio.h>
	
	void baza(int n, int b)
	{ int r=n%b;
	if (n>=b)
	baza(n/b,b);
	cout<<r;
	}
	
	void main()
	{ int n, b;
	clrscr();
	cout<<" n = ";cin>>n;
	do
	{ cout<<" baza = ";cin>>b;
	}while((b<2) && (b>10));
	cout<<n<< "→"<<baza(n,b);
	}

8. INVERSAREA UNUI CUVÂNT

Să se inverseze caracterele unui cuvânt citit de la tastatură, literă cu literă până la întâlnirea caracterului *spațiu*.

Rezolvare:

Se scrie un subprogram fără parametri formali, *invers()/invers*. La fiecare autoapel, se citește un nou caracter ***c***. Condiția de oprire a lanțului de apeluri este îndeplinită când caracterul citit este *spațiu*. Afișarea caracterelor în ordine inversă este realizată printr-o operație de scriere (*write/cout*). Când este îndeplinită condiția de oprire, caracterele depuse în stivă la fiecare autoapel sunt extrase și afișate (în ordine inversă).

LIMBAJUL PASCAL	LIMBAJUL C/C++

procedure inver;
var c:char;
begin
readln(c);
if (c<>' ') then inver;
write(c);
end;
 	
begin
inver;
end.
	#include<iostream.h>
	
	void inver()
	{ char c;
	cin>>c;
	if (c!='0')
	{inver();
	cout<<c;
	}
	}
	void main()
	{
	inver();
	}

PROBLEME PROPUSE

1. Se consideră următorul program :

LIMBAJUL PASCAL	LIMBAJUL C/C++

var x,y:integer;
 	
procedure t(x:integer; y:integer);
begin
if x>0 then
begin
x:=x-1; y:=y+1; t(x,y);
end;
end;
begin
x:=3;y:=1;
write(x,' ',y,' ');
t(x,y);
writeln(x,' ',y);
end.
	#include <iostream.h>
	int x,y:integer;
	void t(int x, int y)
	{ if (x>0)
	{ x = x-1 ; y = y+1;
	t(x,y);
	}
	}
	
	void main()
	{ x = 3; y = 1;
	cout<<x<<" "<<y<<" ";
	t(x,y);
	cout<<x<<" "<<y<<" "<<endl;
	}

Precizați ce va tipări programul de mai sus:

a) 3 1 1 1; **b)** 3 1 0 3; **c)** 3 1 0 2; **d)** 3 1 3 1; **e)** 3 1 0 1.

Scrieți modificările în program astfel încât după execuție să se tipărească 3 1 0 1.

Scrieți modificările în program astfel încât după execuție să se tipărească 3 1 0 4.

LIMBAJUL C/C++
tream.h>

MBAJUL C/C++
eam.h>
y)
y = y+1;
<<y<<" ";
<<y<<" "<<endl;

e) 3 1 0 1.
e tipărească 3 1 0 1.
e tipărească 3 1 0 4.

2. Scrieți subprograme recursive pentru calculul valorii următoarelor polinoame care verifică relațiile de recurență, într-un punct x real, cunoscut.

1. Polinomul Cebâșev de speța I

$$T_n(x) = \begin{cases} 1, & \text{dacă } n = 0 \\ x, & \text{dacă } n = 1 \\ 2xT_{n-1}(x) - T_{n-2}(x), \end{cases}$$

2. Polinomul Cebâșev de speța II

$$U_n(x) = \begin{cases} 1, & \text{dacă } n = 0 \\ 2x, & \text{dacă } n = 1 \\ 2xU_{n-1}(x) - U_{n-2}(x), \end{cases}$$

3. Polinomul lui Legendre

$$(n+1)P_{n+1}(x) = \begin{cases} 1, & \text{dacă } n = 0 \\ x, & \text{dacă } n = 1 \\ (2n+1)xP_n(x) - nP_{n-1}(x), & \text{dacă } n = 0 \end{cases}$$

3. a) Să se stabilească care este efectul următorului subprogram recursiv pentru apelul s(4):

VARIANTA PASCAL/BORLAND PASCAL	VARIANTA C++/C

procedure s(y: integer) ;
begin
if (y>0) then
begin
writeln('***');
s(y-1);
writeln('aaa');
end
end;
	void s(int y)
	{ if (y>0)
	{ cout<<"***"<<endl;
	s(y-1) ;
	cout<<"aaa"<<endl;
	}
	}

b) Modificați subprogramul și precizați apelul din programul principal/funcția principală astfel încât să se afișeze :

**aa
**aa
**aa
**aa

4. Se dă următoarea funcție recursivă care calculează valoarea polinomului lui Hermite în punctul x real. Cerințe:

a) Scrieți expresia după care se calculează valoarea polinomului, punând în evidență recurența matematică.

VARIANTA PASCAL/BORLAND PASCAL	VARIANTA C++/C

function her(x:real; n: integer) : real;
begin
if (n=0) then her:=1
else
if (n=1) then her :=2*x
else
her:=2*x*her(x,n-1)- 2*(n-1)*her(x,n-2);
end;
	float her(float x, int n)
	{ if (n==0) return 1;
	else
	if (n==1) return 2*x;
	else
	return 2*x*her(x,n-1) - 2*(n-1)*her(x,n-2);
	}

b) Care este valoarea afișată pentru apelul her(1,3):

(i) –4; (ii) 2; (iii) 0; (iv) 8; (v) 4.

c) Să se scrie un subprogram iterativ echivalent cu cel dat.

5. Se consideră următorul subprogram:

Varianta Pascal/Borland Pascal	Varianta C++/C

a) procedure u(k: integer);
var d,p:integer;
begin
if k>1 then
begin
d:=s+1; p:=0;
while (k mod d =0) then
begin
k:=k div d;
p:=p+1
end;
write(d,' ',p,' ');
s:=d;
u(k)
end
else writeln(k);
end;
	void u(int k)
	{ int d,p;
	if (k>1)
	{ d =s+1; p=0;
	while (k % d = =0)
	{ k =k div d;
	p =p+1
	}
	cout<<d<<" "<<p<<" ";
	s = d;
	u(k);
	}
	else cout<<k<<endl;
	}

a) Ce se va afișa pentru n=18 și s=1 ?

(i) 2 1 9 3 2 1; (ii) 2 1 3 2 1; (iii) 18 2 1 3 2 1; (iv) 2 1 9 3 1; (v) 2 1 3 1 1.

b) Scrieți instrucțiunea sau instrucțiunile necesare apelului acestui subprogram, precizând limbajul utilizat.

c) Formulați un enunț pe baza căruia să poată fi aplicat acest subprogram.

6. Fie șirul de numere definit astfel:

$$a_{n+1} = \begin{cases} m, dacă\ n = 0 \quad m \in R \\ \frac{a_n}{4}, \text{ dacă n este par} \\ 4a_n - 1, \text{ altfel} \end{cases}$$

a) Să se scrie un subprogram recursiv pentru calculul termenului $n \in N$, n>0, pentru o valoare oarecare a parametrului real *m*.

b) Care este valoarea termenului a_5, dacă m=2:

(i) 5; (ii) 6; (iii) 12; (iv) 7; (v) 2.

7. Fie următorul subprogram recursiv:

VARIANTA PASCAL	VARIANTA C++

procedure **u**(a:integer)
begin
if a **div** 8 > 0 then u(a **div** 8);
write(a **mod** 8)
end;
	void **u**(int a)
	{ if (a/8 >0) u(a div 8);
	cout<<a%8;
	}

a) Pentru apelul **u**(39) precizați : numărul de autoapeluri ce se va afișa la revenirea din subprogram:

(i) 2 / 74; (ii) 1 / 74; (iii) 2 / 47; (iv) 1 / 47; (v) 3 / 47.

b) Transformați subprogramul recursiv într-un subprogram iterativ, precizându-se varianta de limbaj utilizată.

8. Se dă următorul subprogram recursiv pentru calculul valorilor unei funcții:

VARIANTA PASCAL	VARIANTA C++

function f(x:integer):integer;
begin
if x >= 15 then f:=x-2
else f:=f(x+4)
end;
	int **f**(int x)
	{ if (x >15) return x-2;
	else return f(x+4) ;
	}

a) Să se identifice care este varianta corectă a conținutului stivei pentru calculul valorii funcției în punctul x=5 și care este valoarea funcției.

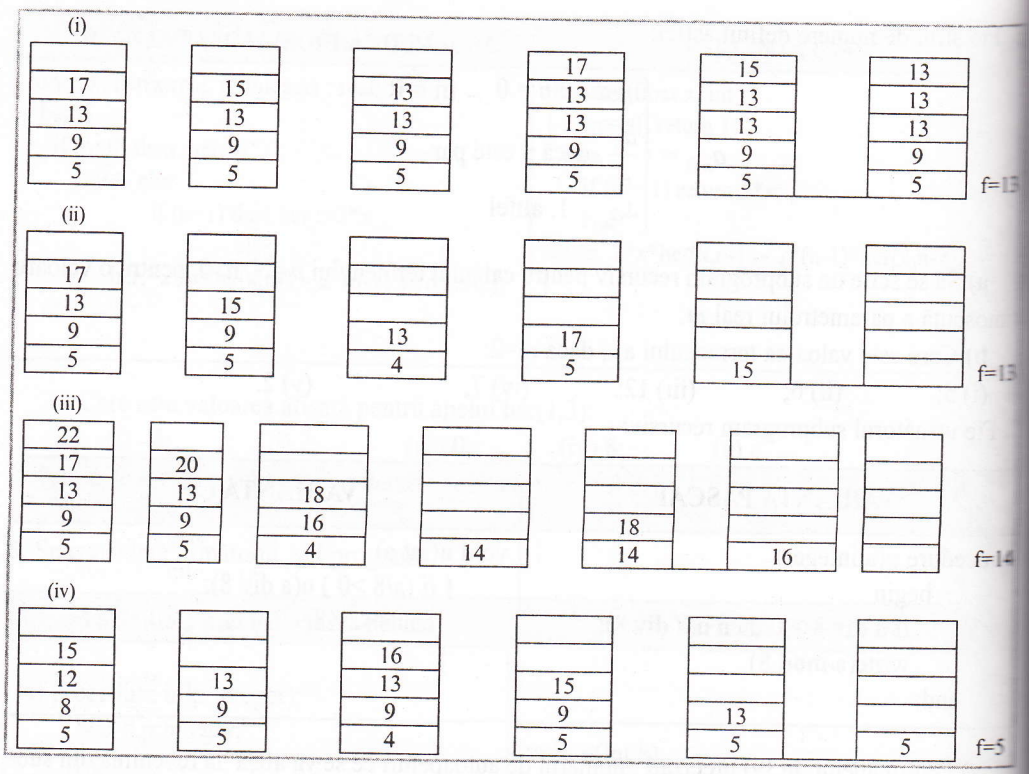


Figura 47.

b) Care sunt valorile posibile pe care le poate lua **f** ?
(i) {13,14,15,16}; (ii) {12,13,15,16}; (iii) {12, 14,16,18}; (iv) {14, 15, 16,18}
(v) o infinitate de valori.

9. Implementați recursiv preluările specifice structurii dinamice de tip *coadă*.
10. Implementați recursiv prelucrările specifice structurii dinamice de tip *stivă*.
11. Explicați aspectul recurent al structurilor dinamice arborescente (arborele de familie).